ON THE DETECTION OF NEGATIVE CYCLES IN A GRAPH

A THESIS

Presented to

The Faculty of the Division

of Graduate Studies

By

Dennis P. Shea

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Operations Research

Georgia Institute of Technology

May, 1977

ON THE DETECTION OF NEGATIVE CYCLES

IN A GRAPH

Approved:

John J. Jarvis, Chairman

[illegible] T. Mentsch

Robert G. Parker

Date approved by Chairman: 12 May 77

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Dr. John J. Jarvis who served as my thesis advisor. Despite his busy schedule, Dr. Jarvis always found time to review my work and offer helpful suggestions.

Special thanks are also due to Dr. S. J. Deutsch and Dr. R. G. Parker, the members of my reading committee, for their continued support and helpful recommendations.

I wish to thank Mrs. Sharon Butler who typed the final draft of this thesis.

Finally, I especially wish to thank my wife, Amee, who typed the first two drafts of this manuscript. Without her understanding and support throughout the many lost evenings and weekends, this thesis would not have been completed.

This work was supported by Grant Number 75NI-99-0091, awarded by the Law Enforcement Assistance Administration, U. S. Department of Justice, under the Omnibus Crime Control and Safe Streets Act of 1968, as amended. Points of view or opinions stated in this document are those of the author and do not necessarily represent the official position or policies of the U. S. Department of Justice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

SUMMARY

The thesis discusses negative cycles and their applications to network flow problems. Four algorithms are identified to locate negative cycles in a graph. An experimental design is employed to evaluate the effects of algorithm, nodes, density, and arc distribution in detecting negative cycles on random networks. Extensive analysis is performed on (1) the computational time to detect a negative cycle, (2) the quality of the negative cycle, (3) the sum of the arc costs around the cycle, and (4) the number of nodes in the cycle.

The results exhibit the Yen algorithm to be the most reliable in terms of locating negative cycles of high quality in low computational time. Both theoretical and empirical evidence indicate that the direct search method of Florian and Robert, while generally requiring the least computational time, may be limited in application due to its immense computational upper bound.

CHAPTER I

INTRODUCTION

The Negative Cycle Approach to Network Flow Problems

Network analysis is the branch of Operations Research which deals with problems concerning the movement, whether actual or conceptual, of people and/or commodities. Network analysis has long played an important role in electrical engineering. Recently, there has been a growing awareness that certain concepts and tools of network theory are also very useful in many other contexts, such as transportation systems and production scheduling.

According to the theory of graphs (networks), a "network" consists of a set of junction points called "nodes" with certain pairs of the nodes being joined by lines called "arcs." A flow of some type is considered to circulate among the nodes via the arcs. As an example of a "physical" network consider the highway system where intersections can be considered as nodes, roads can be considered as arcs and the flow is comprised of vehicles.

Minimal Cost Flows

A fundamental problem in network theory concerns sending flow between specified nodes such that the cost of sending the flow is a minimum and the flow is within the

lower and upper bounds of each arc in the network.

The mathematical formulation of this problem, where the flow is to be sent from node 1 (source) to node N (sink), is as follows:

The minimal cost flow problem is to find a flow X of value v which minimizes the quantity

$$Q(X) = \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij}x_{ij}$$

and satisfies

1. $0 \leq x_{ij} \leq u_{ij} \qquad i = 1,2,\ldots,N \qquad j = 1,2,\ldots,N$

2. $$\sum_{j=1}^{N} (x_{1j}-x_{j1}) = v$$

3. $$\sum_{j=1}^{N} (x_{ij}-x_{ji}) = 0, \qquad i = 2,\ldots,N-1$$

4. $$\sum_{j=1}^{N} (x_{Nj}-x_{jN}) = -v$$

where u_{ij} is the capacity along arc (i,j) and c_{ij} is the cost associated with sending a unit of flow along arc (i,j). Constraints (2)-(4) are conservation of flow equations and state that the flow may be neither created nor destroyed in the network. Constraint (2) indicates that the net flow leaving the source node must equal v units while constraint (4) indicates that the net flow entering the sink node must

equal v units. For the remaining nodes, the sum of the flows entering the node must equal the sum of the flows leaving the node.

In some problems, c_{ij} may be regarded as the distance between nodes i and j. These distances need not be symmetric, i.e. the distance from i to j need not equal the distance from j to i. In such problems, we wish to send v units of flow from the source to the sink, such that the distance traveled is a minimum. We shall use both cost and distance interchangeably throughout this thesis.

Many practical problems such as transportation, transhipment, and assignment can be formulated as special cases of the minimal cost flow problems.

Numerous algorithms have been proposed to solve these problems. These algorithms may be classified as (1) primal algorithms and (2) dual algorithms.

Dual methods attempt to add one unit of flow to a network by solving a shortest path problem on a marginal cost network. The primal methods start with a feasible flow of v units and attempt to minimize the cost of sending v units from the source to the sink by reassigning portions of this flow around negative cycles. This contrasts the dual methods in which feasible flows are not available until the computations terminate.

Negative Cycles in Network Flows

Busacker and Saaty [6] have proven a powerful theorem

concerning negative cycles on which primal algorithms for network flow problems are based.

Theorem I. X is a minimal cost flow of value v, if and only if, there are no directed cycles of negative length in the marginal cost network.

A negative cycle is a sequence of distinct directed arcs of the form $[(i_0, i_1), (i_1, i_2), \ldots, (i_q, i_0)]$ involving distinct nodes such that the sum of the costs associated with the arcs is negative. In this thesis we shall be primarily concerned with locating negative cycles in a network.

A "physical" example of a negative cycle can be found in the exchange of foreign currencies. Due to differences in the buying and selling rates it is possible to start with X dollars in American currency, exchange this for a foreign currency, and follow a pattern of exchanging the present currency for a distinct foreign currency finally selling a foreign currency for Y dollars in American currency where $Y < X$. Fig. 1-1 is a graphical representation of this example.

The marginal cost network G(X) is defined with respect to an integer primal feasible set of flows in the original network. The marginal cost network contains all of the nodes from the original network and arcs as follows:

$$(i,j), \quad \text{if } x_{ij} < u_{ij} \quad \text{and} \quad x_{ji} = 0$$

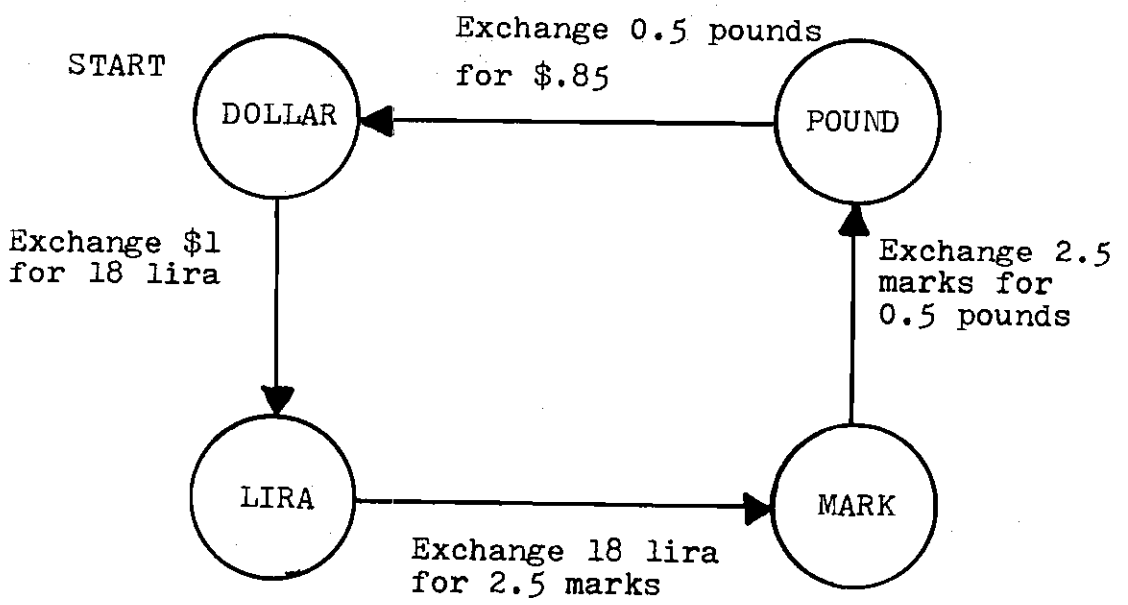


Figure 1-1. Example of a Negative Cycle in the Exchange of Foreign Currency

$$(j,i), \quad \text{if} \quad x_{ij} > 0$$

with revised capacities:

$$u'_{ij} = u_{ij} - x_{ij}, \quad \text{if} \quad x_{ij} < u_{ij} \quad \text{and} \quad x_{ji} = 0$$

$$u'_{ji} = x_{ij}, \quad \text{if} \quad x_{ij} > 0$$

and with revised arc costs:

$$c'_{ij} = c_{ij}, \quad \text{if} \quad x_{ij} < u_{ij} \quad \text{and} \quad x_{ji} = 0$$

$$c'_{ji} = -c_{ij}, \quad \text{if} \quad x_{ij} > 0$$

Utilizing the above theorem, an algorithm for the minimal cost flow problem becomes obvious.

A Minimal Cost Flow Algorithm

1. Find a feasible flow from node 1 to node N of value v utilizing the Ford and Fulkerson maximum flow routine [16].

2. Construct the marginal cost network G(X) as described in the previous section and search G(X) for the existence of a negative cycle. A negative cycle indicates a change in the flows around the cycle, which will result in a reduction in the total cost. If there does not exist any negative cycles in the marginal cost network, the

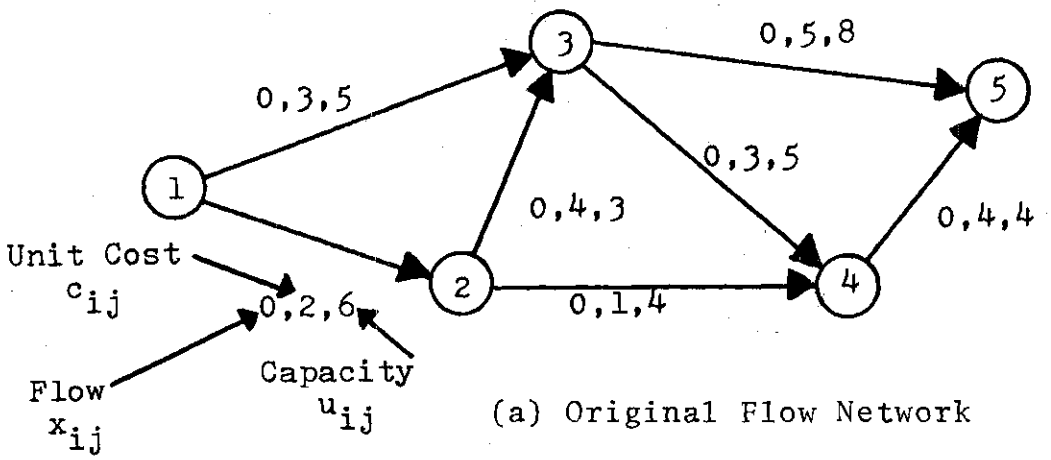
present flow being tested is optimal.

3. If a negative cycle is found an improved flow (one with lower total cost) can be obtained by sending a positive unit flow around the cycle. Sending a unit of flow from node j to node i along arc (i,j) corresponds to decreasing the flow by one unit in arc (i,j). The value of the flow v will remain unchanged.

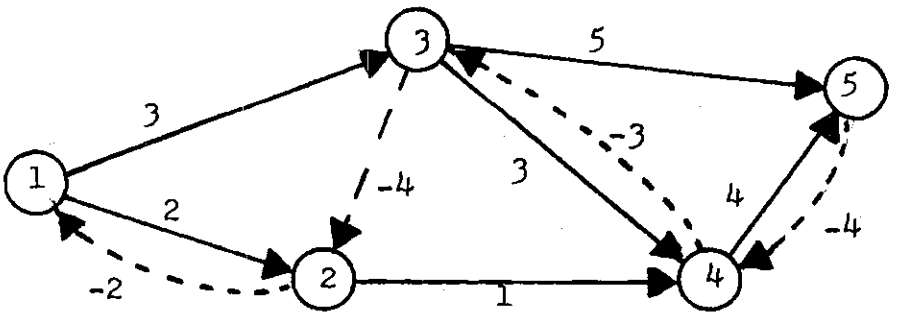
4. After changing the flows around the cycle, return to step 2 and construct the new marginal cost network. Continue until there are no longer any negative cycles in the marginal cost network.

As an example of the way in which a marginal cost network is created, consider the network in Fig. 1-2(a). The ordered triple (x_{ij}, c_{ij}, u_{ij}) represents the flow on arc (i,j), the cost of sending a unit of flow along arc (i,j) and the capacity of arc (i,j), respectively. Suppose 3 units of flow were sent from the source to the sink via the path (1,2), (2,3), (3,4), (4,5). The cost of sending this flow is 39. The associated marginal cost network is shown in Fig. 1-2(b). Note that the "forward" arc (2,3) is missing from the marginal cost network since the arc is saturated, i.e. $x_{23} = u_{23}$. The "reverse" (dashed) arcs correspond to arcs having positive flow, which could be cancelled--by using the arc in the reverse direction--without destroying feasibility.

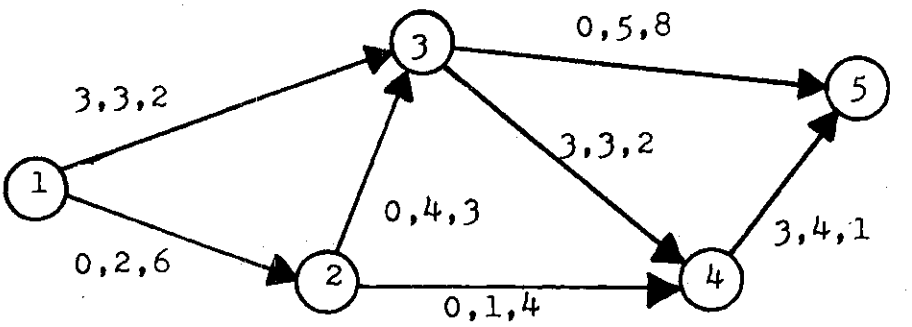
The marginal cost network contains a negative cycle passing through nodes 1, 3, 2, and 1 in sequence. Actually,



(a) Original Flow Network



(b) Marginal Cost Network



(c) Updated Flow Network

Figure 1-2. Example of a Marginal Cost Network

the network contains three negative cycles but the algorithm requires only one.

We can improve the objective function while maintaining a feasible solution by sending flow around this cycle. The value of the cycle or the sum of the costs around the arcs in the cycle is -3 and indicates the potential reduction in the objective function by sending a unit of flow around the cycle. The minimum capacity of any arc in the cycle is 3 corresponding to the "reverse" arcs (3,2) and (2,1). Therefore, the largest amount of flow which can be sent around the cycle is 3 units. The updated network is shown in Fig. 1.2(c). The cost of sending this improved flow is 30.

Applying the Negative Cycle Approach to the Assignment Problem

The assignment problem is to fill N jobs by as many men at least total cost. Let c_{ij} represent the cost of using man i in job j, then a mathematical statement of the problem is to find a permutation matrix $X = (x_{ij})$ of order N, which minimizes the total cost $Q(X) = \Sigma x_{ij} c_{ij}$, where $x_{ij} = 1$ implies that man i (i = 1, 2, ..., N) is assigned to job j (j = N+1, N+2, ..., 2N).

The equivalent minimum cost flow problem is illustrated for the case N=3 in Fig. 1-3.

An Algorithm for the Assignment Problem

1. The algorithm can be started with any feasible

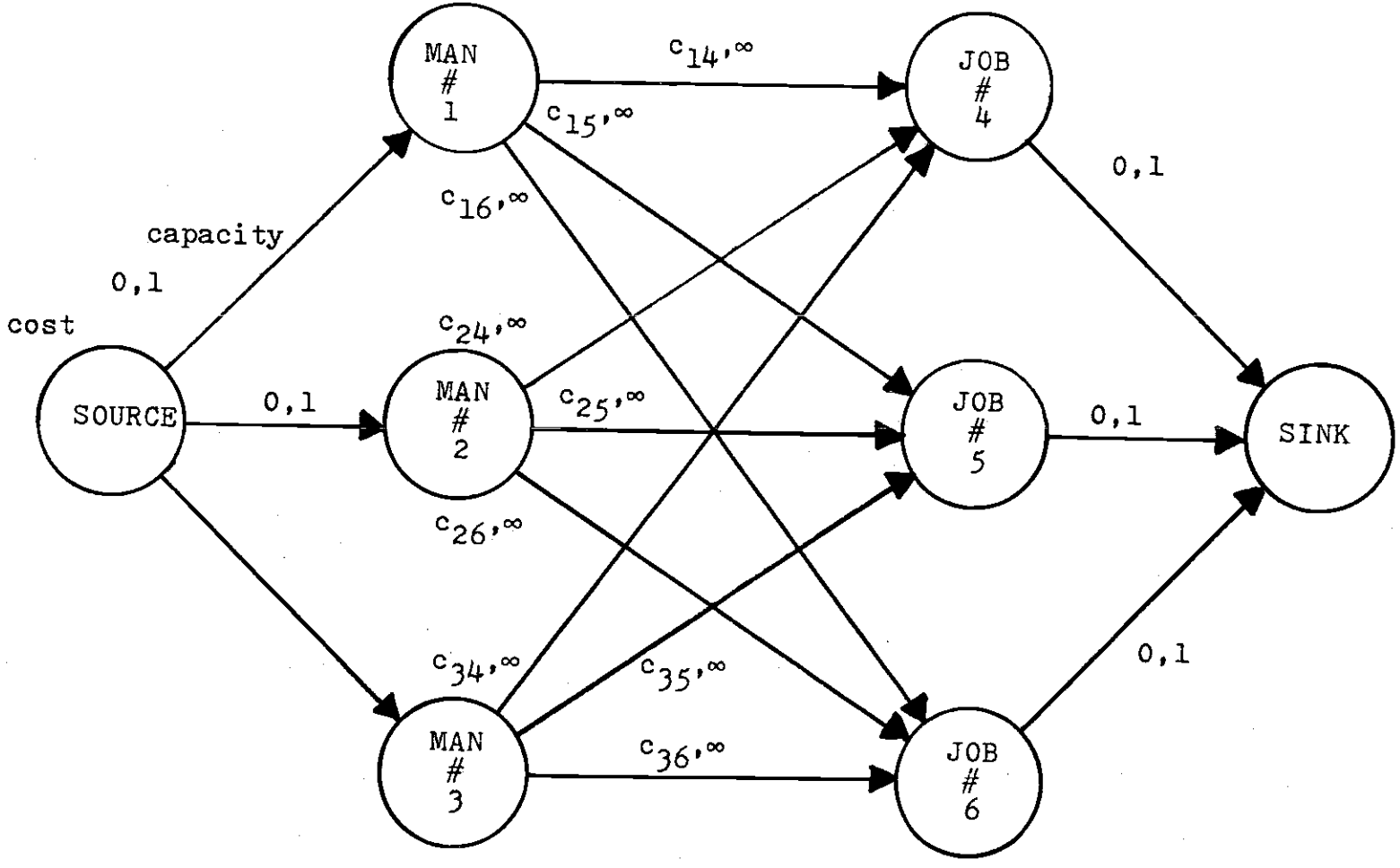


Figure 1-3. Network Representation of an Assignment Problem

solution. This corresponds to finding a flow of value v=N. Let X be a trial solution and let C'(X) be the (2N)x(2N) matrix with rows and columns each corresponding to both the men and jobs of the problem and with elements

$$c'_{ij} = \infty, \; i,j = 1, \ldots, N \quad \text{or} \quad i,j = N+1, \ldots, 2N$$

$$= c_{ij}, \; i = 1, \ldots, N; \quad j = N+1, \ldots, 2N$$

$$= -c_{ji}, \; x_{ji} > 0 \quad i = N+1, \ldots, 2N; \; j = 1, \ldots, N$$

$$= \infty, \; x_{ji} = 0 \quad i = N+1, \ldots, 2N; \; j = 1, \ldots, N$$

2. Test for the existence of a negative cycle in the matrix C'. An improved solution will result by sending a unit of flow around the negative cycle. The nodes in the negative cycle will alternate between men and machines (jobs), thus each arc will correspond to the assignment (removal) of a man to (from) a job.

As an example consider the situation with four men i = 1, 2, 3, 4 to be assigned to four jobs j = 5, 6, 7, 8. Suppose the trial solution was $X_{16} = X_{27} = X_{38} = X_{45} = 1$. Further, suppose the following negative cycle was located in the C' matrix (7,2), (2,5), (5,4), (4,7). Then an improved solution to the assignment problem will result with $X_{16} = X_{25} = X_{38} = X_{47} = 1$. Recreate the marginal cost matrix

using the updated trial solution and test for negative cycles. When there are no negative cycles in the C' (marginal cost) matrix the optimal solution is at hand.

Applying the Negative Cycle Approach to the Transportation Problem

The algorithm presented for solving the transportation problem utilizing negative cycles is similar to the one presented for the assignment problem since each of these problems is a special case of the other.

The transportation problem for the single commodity case can be defined as follows:

We wish to ship a single commodity from n plants $P_1, P_2, \ldots, P_n$, with capacities $a_1, a_2, \ldots, a_n$ to m warehouses $W_{n+1}, \ldots, W_{n+m}$ with requirements $b_{n+1}, \ldots, b_{n+m}$. If c_{ij} represents the cost of shipping a unit from P_i to W_j, and x_{ij} the quantity shipped from P_i to W_j, the problem is to minimize the total cost

$$Q(X) = \sum_{j=n+1}^{n+m} \sum_{i=1}^{n} x_{ij} c_{ij}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = b_j, \quad j = n+1, n+2, \ldots, n+m$$

$$\sum_{j=n+1}^{n+m} x_{ij} = a_i, \quad i = 1, 2, \ldots, n$$

$$x_{ij} = 0, 1, \ldots$$

where we assume that the a_i's and the b_j's are positive integers and $\Sigma a_i = \Sigma b_j$.

As previously mentioned, the assignment problem and the transportation problem have a similar structure. The essential difference is that in the assignmnet problem both the right-hand-side vector of the constraint matrix and the capacities on each arc are restricted to equal one. These restrictions are lifted in the transportation problem. Therefore, when a negative cycle is located in the marginal cost network, it may be possible to send more than a single unit of flow around the cycle to obtain an improved solution.

Convex-Cost Networks

The foregoing applications involved linear cost functions. The negative cycle approach can be extended to convex-cost networks by using marginal costs to compute c'_{ij} in C'.

If we represent each arc cost function by p_{ij}, the minimal cost flow problem is to find the minimum of

$$Q(X) = \sum_i \sum_j p_{ij} (x_{ij})$$

subject to

$$\sum_j x_{ij} = a_i \qquad i = 1, \ldots, N$$

$$\sum_i x_{ij} = b_j \qquad j = 1, \ldots, N$$

where a_i is the supply at node i and b_j is the demand at node j.

Then the marginal costs c'_{ij} are given by

$$c'_{ij} = [p_{ij}(x_{ij}+1) - p_{ij}(x_{ij})] \quad \text{if} \quad x_{ij} < u_{ij} \quad \text{and} \quad x_{ji} = 0$$

$$c'_{ji} = [p_{ij}(x_{ij}-1) - p_{ij}(x_{ij})] \quad \text{if} \quad x_{ij} > 0$$

where u_{ij} is the capacity of arc (i,j).

Statement of the Problem

Clearly, the efficiency of the negative cycle approach in solving network problems is strongly dependent on the efficiency in detecting and tracing negative cycles in a graph. Due to the number of marginal cost networks which must be created and examined for negative cycles, if an efficient scheme cannot be found it would seem that the algorithm would become impracticable and the dual methods our only recourse. It is the intent of this thesis to prove the superiority of the negative cycle approach to solving

network problems. To accomplish this we must show that the location and subsequent tracing of negative cycles can be handled in an efficient manner.

Outline of the Thesis

In this chapter we have attempted to develop the motivation for identifying efficient techniques to locate negative cycles. The remaining chapters of this thesis outline several approaches for detecting negative cycles in a graph, and present both theoretical and experimental analysis of the efficiency of each of these algorithms.

Chapter II describes several important algorithms existing in the literature for locating negative cycles. Chapter III presents a theoretical analysis of the efficiency of the algorithms. The approach is conservative in that it describes the efficiency on a "worse case" problem. Chapter IV outlines a design to test experimentally the efficiency of the algorithms presented in Chapter II. Chapters V and VI discuss experimental results carried out under the auspices of the design. An application of the negative cycle approach to a particular flow problem is presented in Chapter VII, while Chapter VIII summarizes the results of this thesis and suggests extensions of the work.

CHAPTER II

THE ALGORITHMS

The algorithms for locating and tracing negative cycles in a graph can be classified as:

1. Shortest Path Algorithms
2. Direct Search Methods

The shortest path algorithms attempt to find the shortest routes in a graph. If the graph does not contain negative cycles these algorithms produce a "legitimate" set of shortest routes. Otherwise, the occurrence of a "shortest path" containing more than N-1 arcs (N = the number of nodes) or a negative length route from a node in the graph to itself, indicates the existence of a negative cycle. The shortest path is traced and serves to locate the nodes in the cycle.

The direct search method identifies a unique property possessed of negative cycles and exploits this property in locating and tracing the cycles.

Literature Search

Many of the important algorithms for detecting negative cycles in a graph concern locating shortest paths. Given a graph containing N nodes, a path between any pair of nodes may contain at most N-1 arcs. If a shortest path can be

found which contains more than N-1 arcs, then an arc must be repeated and a negative cycle formed. To understand the negative cycle algorithms we therefore need to understand the shortest path problem.

The shortest path problem is defined on a set of N nodes, numbered arbitrarily from 1 to N, and the N x N matrix C, not necessarily symmetric, whose element c_{ij} represents the length of the directed arc connecting node i to node j. The problem is to find the path of shortest length connecting a specified node to all remaining nodes.

One of the first computationally efficient schemes for solving this problem was presented by Bellman [3]. Bellman reports that up until that time (1958), the shortest path procedures were primarily enumeration of all possible paths. Since the number of paths is finite, the problem reduces to choosing the smallest from a finite set of numbers. A network containing N nodes contains (N-1)! sets of shortest paths from the origin to every remaining node. Therefore, complete enumeration becomes infeasible for networks consisting of more than a few nodes.

Bellman proposed a dynamic programming algorithm for the problem when the associated arc distances are nonnegative.

By the principle of optimality, π_j (the optimal distance from node 1 to node j) must satisfy the nonlinear system of equations

$$\pi_j = \min_{i \neq j} [c_{ij} + \pi_i] \quad j = 2, 3, \ldots, N$$

$$\pi_1 = 0$$

Bellman suggested the following iterative algorithm

$$\pi_j^{(o)} = c_{1j} \quad j = 1, 2, \ldots, N$$

$$\pi_j^{(k+1)} = \min_{i \neq j} (c_{ij} + \pi_i^{(k)}) \quad j = 2, 3, \ldots, N$$

$$\pi_1^{(k+1)} = 0$$

$$k = 0, 1, 2, \ldots$$

$\pi_j^{(k)}$ represents the minimum distance for a path passing through at most k intermediate nodes. Termination with the optimal solution occurs when $\pi_j^{(k)} = \pi_j^{(k-1)}$, $j = 1, 2, \ldots, N$. Convergence is guaranteed in no more than N-2, i.e. $k = N-2$, iterations since no path contains more than N-1 arcs.

Ford and Fulkerson [16] presented a similar algorithm which extended to the more general problem where some c_{ij} are negative. When the procedure is applied to a problem with some negative distances c_{ij}, either convergence will occur on or before the (N-1)st iteration, indicating no negative

cycles exist and the solution is optimal; or a change in some π_j will occur on the Nth iteration, indicating the existence of a negative cycle.

The most efficient algorithm for the shortest path problem when the arc distances are restricted to be nonnegative is the Dijkstra algorithm [8]. The Dijkstra algorithm, also a dynamic programming algorithm, permanently labels at least one node during each iteration. Consequently, it requires at most N-1 iterations to determine the shortest path from the origin to all remaining nodes. The computational requirements are N(N-1)/2 additions and N(N-1) comparisons to solve the problem compared to the Ford-Fulkerson and Bellman algorithms, which could require as many as N^3 additions and comparisons.

Bazaraa and Langley [2] proposed an algorithm to convert a distance matrix with negative elements into a nonnegative distance matrix. The Dijkstra algorithm can then be applied to the resultant matrix to determine the shortest paths. If the distance matrix cannot be transformed into a nonnegative matrix, a negative cycle exists in the corresponding network. The computational upper bound (although rarely achieved) on this approach is $3N^3$.

By processing the nodes alternately forward and backwards, and minimizing only over nodes previously treated, Yen [29], [30], [32] has produced a dynamic programming algorithm which reduces the amount of computational effort to

$N^3/2$. This number is half that required by the original dynamic programming algorithms of Ford-Fulkerson and Bellman. This algorithm also indicates a negative cycle when a node label (functional equation) changes value on the last iteration.

Dantzig proposed the first simplex algorithm for the shortest path problem [7]. Bennington [4] refined this algorithm to show that the set of basic feasible solutions could be restricted to the set of arborescences centered at the origin. An arborescence centered at the origin consists of N-1 arcs which form a unique path from the origin to all remaining nodes without forming any cycles. Bennington also introduced a test for negative cycles into the simplex algorithm.

Klein [25], in his paper on the negative cycle approach to the minimal cost flow problem introduced a matrix approach to locate negative cycles. This algorithm, credited to Hu [22], finds the shortest distance between every pair of nodes. A negative cycle is indicated when a shortest path from a node to itself is found which is negative. The algorithm produces more information than is required to find the negative cycle and its computational upper bound is of the order N^3.

The only direct search method to locate negative cycles is due to Florian and Robert [13]. The algorithm is based on a property of negative partial sums of finite

sequences. The only disadvantage of this algorithm can be shown to be its excessive computational upper bound.

We restricted our evaluation to the algorithms of Yen, Bennington, Florian, and Ford-Fulkerson. The first three are considered the most efficient algorithms from the fields of dynamic programming, linear programming, and direct search methods, respectively. The Ford-Fulkerson, also a dynamic programming based algorithm, has proven its reliability over time and would act as a "control algorithm" with which to compare the performance of the remaining three algorithms. The following section describes the algorithms in detail including a flowchart depicting the sequence of operations for each algorithm.

Shortest Path Algorithms

As reported earlier, there are several shortest path algorithms available to the practitioner. We have limited this thesis to examining those which are considered to be most efficient.

The Method of Ford and Fulkerson

The Ford-Fulkerson algorithm is a dynamic programming based algorithm for finding all shortest paths from a fixed node. During the algorithm, all nodes x will receive a label of the form $[z,\pi_x]$. The first element of this ordered pair corresponds to the preceding node in a path from the origin to node x. Node x is considered labeled from node z. The

second element of the pair is the distance along this path from the origin to node x.

Initially, the source node is given a label [-,0] while all other nodes are labeled [-,∞]. The node labels are updated by searching for an arc (x,y) such that $\pi_x + c_{xy} < \pi_y$. If such an arc is found the new label on node y becomes $[x, \pi_x + c_{xy}]$. If no such arc is found, the algorithm is terminated and the π_y's are the lengths of the shortest paths from the source node to node y.

The paths can be recovered by tracing back the labels from node y to the source. The algorithm proceeds systematically, where in iteration k, an attempt is made to improve (lower) the node label on node j, j = 1, 2, ..., N, by using the label on node i, i = 1, 2, ..., N, $i \neq j$ and the arc (i,j). The existence of negative cycles in the network is detected when the π_y's at the end of iteration N-1 are not identical to the π_y's at the end of iteration N. A flow chart of the algorithm is presented in Fig. 2-1. An example of this procedure can be found in the appendix.

A proof of the convergence of the Ford-Fulkerson algorithm for the case where the network does not contain any negative cycles is readily available in any "networks" textbook. It would be beneficial at this point to establish the accuracy of the algorithm in the situation where negative cycles exist.

Theorem II. If there exists a π_y at the end of

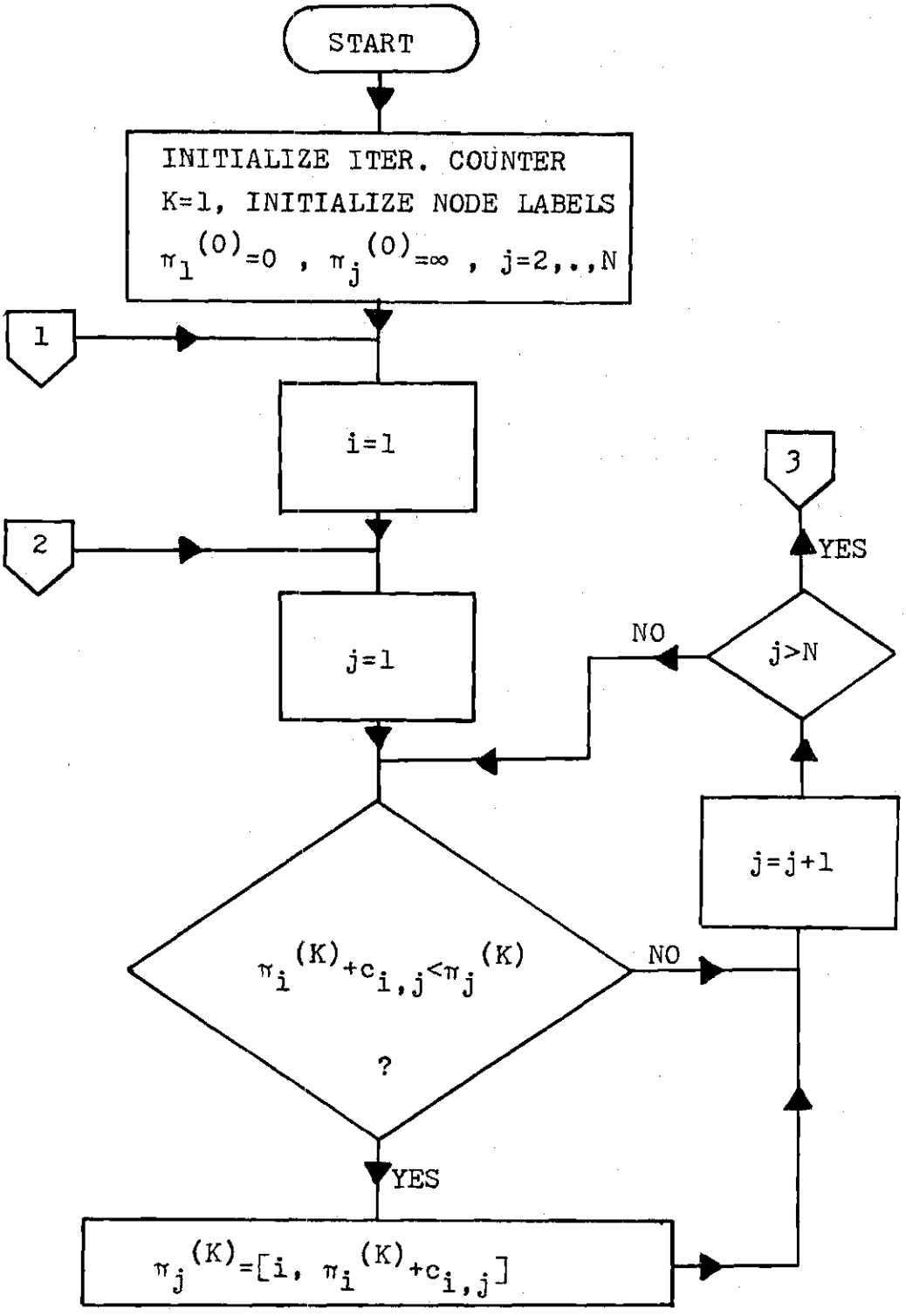


Figure 2-1. Flowchart of the Ford-Fulkerson Algorithm

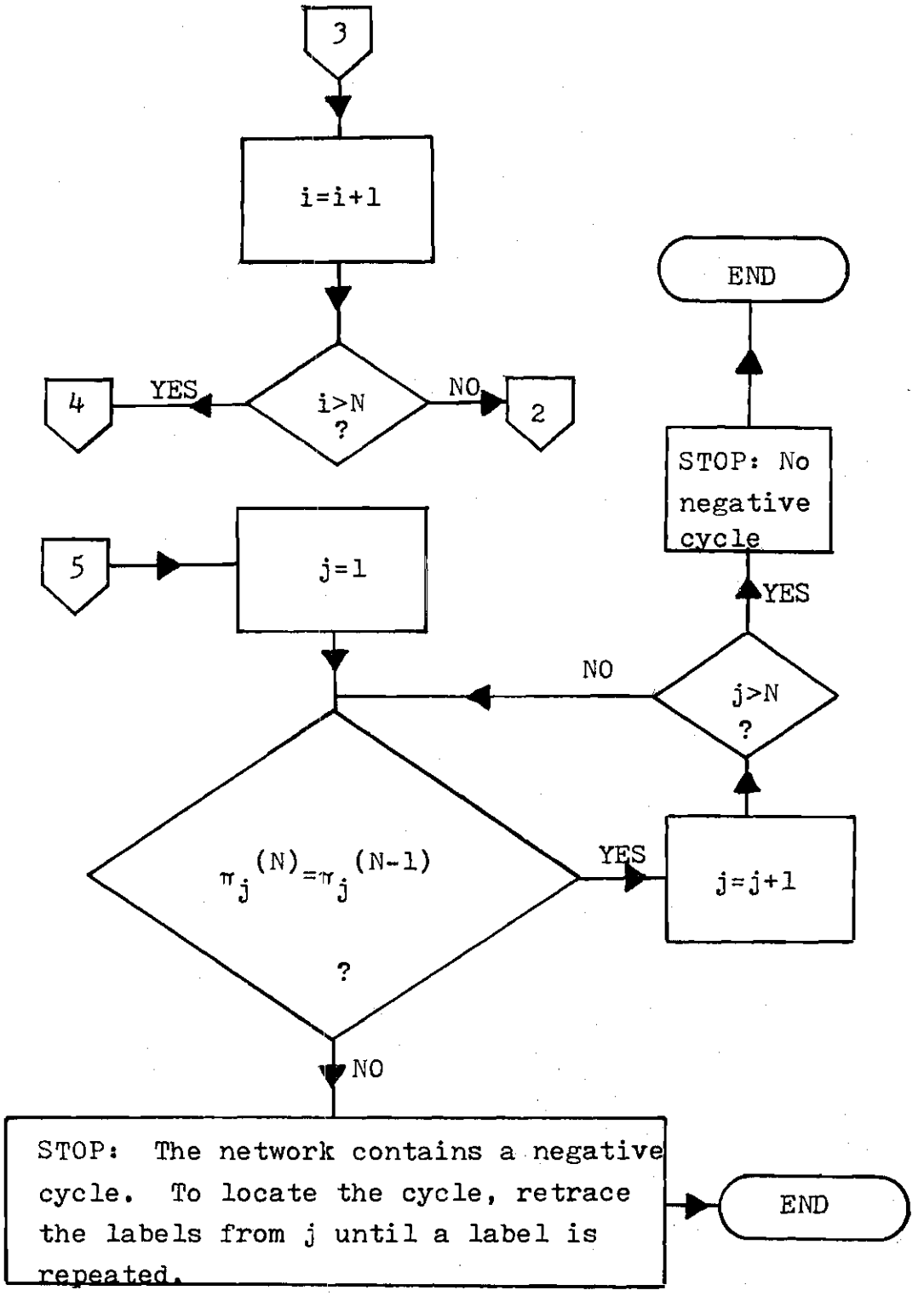


Figure 2-1 (continued)

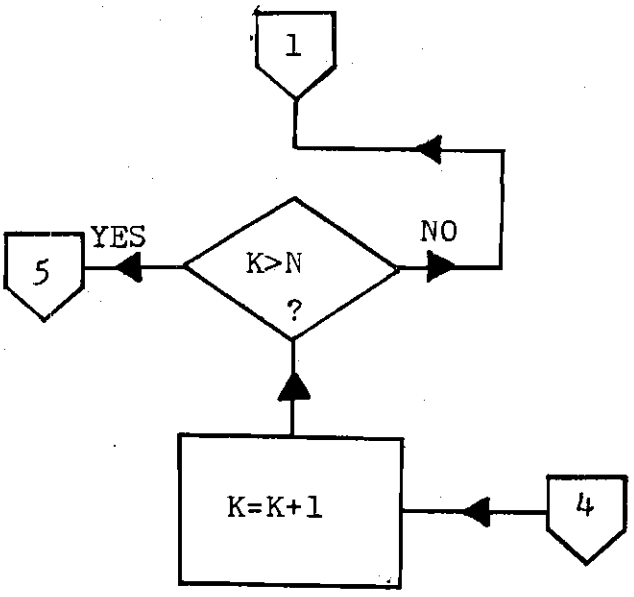


Figure 2-1 (concluded)

iteration N which is not identical to π_y at iteration N-1, then the network contains a negative cycle.

The proof will be divided into two parts. First we shall prove that a cycle exists and then proceed to show that the value of the cycle is negative.

Part I--There Exists a Cycle

Part I will be proven by contridiction. In particular, we shall show that if no negative cycles exist, the algorithm must terminate after at most N-1 iterations. Recall that the algorithm as presented examines the nodes i in sequence $i = 1, 2, \ldots, N$ and attempts to use the label on node i and the arc (i,j) to improve the label on node j, where $j = 1, 2, \ldots, N$; $i \neq j$. Initially, the origin node x is permanently labeled [-,0]. All remaining nodes are temporarily labeled [-,∞]. At the end of the first iteration, all nodes y, whose shortest path from the origin consists of one arc, i.e. (x,y), will be permanently labeled. The permanent labeling will occur when we examine the origin, node x, and the arc, (x,y). If $\pi_x + c_{xy} < \pi_y$, then node y will receive the label $[x, c_{xy}]$. If this is in fact the shortest path then this label will never change during any of the remaining iterations. Next, all nodes z, whose shortest path from the origin consists of two arcs, will become permanently labeled during the second iteration. These nodes z will be labeled from some node y which was permanently labeled during the first iteration, i.e. a node whose shortest path from the

origin consists of the single arc (x,y). The nodes z will become permanently labeled during the second iteration while examining the node y and the arc (y,z). The permanent label on node z would be $[y, \pi_y + c_{yz}]$. Therefore, since the shortest paths can contain at most N-1 arcs, all nodes must be permanently labeled in at most N-1 iterations. The exact number of iterations required is dependent on the sequence in which the nodes are examined. If a node label changes between the N-1st and the Nth iteration, a shortest path must contain more than N-1 arcs. Therefore, some node must be visited twice constituting a cycle. These results are summarized in Fig. 2-2(a) which is the optimal shortest path tree. Fig. 2-2(b) is a "line graph" and is an example of a problem which will require N-1 iterations to solve assuming that the nodes are examined in the sequence 1, 2, ..., N.

Part II--The Value of the Cycle is Negative

The cycle must have originally been formed during the kth iteration where $1 < k \leq N$. The conditions which existed at the beginning of the kth iteration are depicted in Fig. 2-3. At the beginning of iteration k, node i_0 is labeled from node i_x. The cycle will be formed during this iteration when the label on node i_0 is changed to $[i_m, \pi_{i_m} + c_{i_m i_0}]$. Now at the beginning of iteration k, $\pi_{i_1} \geq \pi_{i_0} + c_{i_0 i_1}$. Equality will exist, i.e. $\pi_{i_1} = \pi_{i_0} + c_{i_0 i_1}$ if the label on node i_0 has not changed since it was used to label i_1 during

the previous iteration. If the label on node i_0 has changed, then it must have decreased and $\pi_{i_1} > \pi_{i_0} + c_{i_0 i_1}$. In either case, $\pi_{i_1} \geq \pi_{i_0} + c_{i_0 i_1}$ and similarly

$$\pi_{i_2} \geq \pi_{i_1} + c_{i_1 i_2}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$\pi_{i_m} \geq \pi_{i_{m-1}} + c_{i_{m-1} i_m}$$

$$\pi_{i_0} > \pi_{i_m} + c_{i_m i_0}$$

The last inequality holds as a strict inequality since in order to change the label on node i_0 from i_x to i_m,

$$\pi_{i_0} > \pi_{i_m} + c_{i_m i_0}$$

Bringing the π's to the left side and summing

$$\pi_{i_1} - \pi_{i_0} \geq c_{i_0 i_1}$$

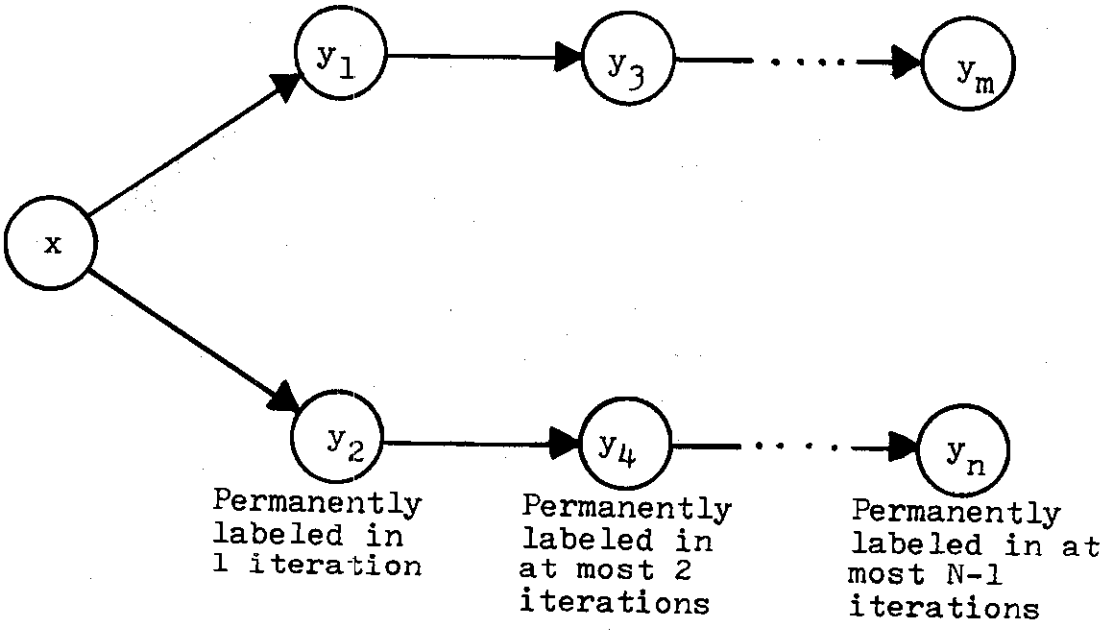
$$\pi_{i_2} - \pi_{i_1} \geq c_{i_1 i_2}$$

$$\vdots \qquad \vdots \qquad \vdots$$

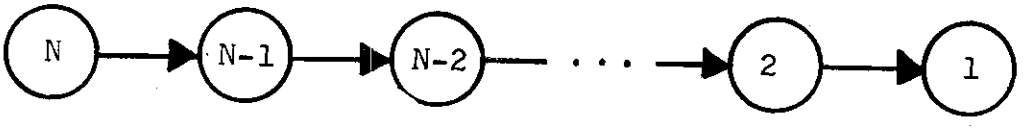
$$\pi_{i_m} - \pi_{i_{m-1}} \geq c_{i_{m-1} i_m}$$

$$\pi_{i_0} - \pi_{i_m} > c_{i_m i_0}$$

$$0 > \Sigma C$$



(a) A Shortest Path Tree



(b) A Line Graph

Figure 2-2. Summary of Ford-Fulkerson Convergence Proof

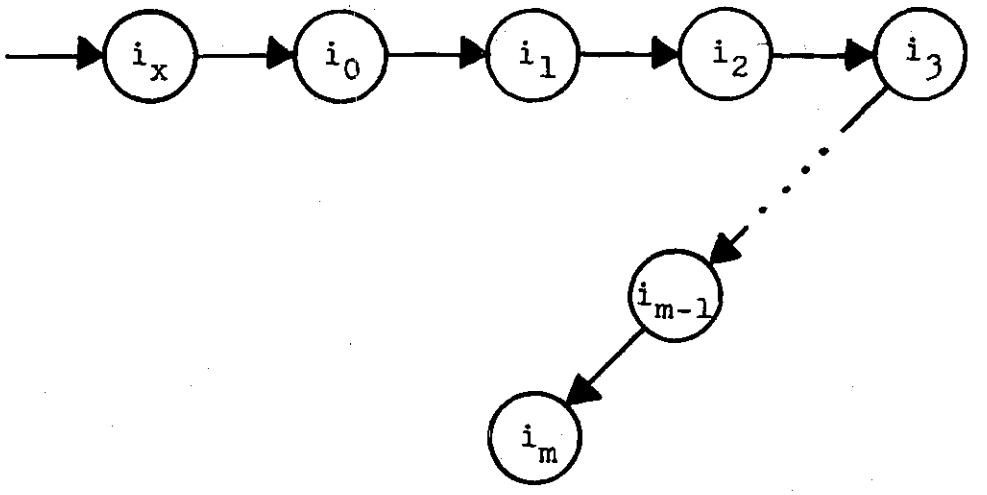


Figure 2-3. Node Labels in the Ford-Fulkerson Algorithm at the Beginning of Iteration k

That is, the sum of the arc costs around the cycle is negative. Q.E.D.

The Algorithm of Yen

This algorithm as originally proposed, determines the shortest routes from the origin to all nodes in N-node general networks. The algorithm is a dynamic programming approach in that a set of functional equations $\pi_i^{(k)}$ are computed during each iteration where $\pi_i^{(k)}$ is the length of the tentative shortest path from node 1 to node N on the kth iteration.

The functional equations are computed using the following iterative procedure:

$$\pi_i^{(0)} = c_{1,i}, \quad i = 1, 2, \ldots, N$$

$$\begin{cases} \pi_i^{(2k-1)} = \min\limits_{1 \le j < i} [\pi_j^{(2k-1)} + c_{ji}, \pi_i^{(2k-2)}], & i = 2, 3, \ldots, N \\ \pi_1^{(2k-1)} = \pi_1^{(2k-2)} \end{cases}$$

$$\begin{cases} \pi_i^{(2k)} = \min\limits_{N \ge j > i} [\pi_j^{(2k)} + c_{ji}, \pi_i^{(2k-1)}], & i = N-1, N-2, \ldots, 1 \\ \pi_N^{(2k)} = \pi_N^{(2k-1)} \end{cases}$$

for $k = 1, 2, \ldots$.

Note that the minimization operator is performed only over those nodes previously processed during the iteration.

The iterative procedure of the algorithm is to be terminated when

$$\pi_i^{(2k)} = \pi_i^{(2k-1)} \quad \text{or} \quad \pi_i^{(2k+1)} = \pi_i^{(2k)}, \; i = 1, 2, \ldots, N.$$

Then $\pi_i^{(2k)}$ or $\pi_i^{(2k+1)}$, $i = 1, 2, \ldots, N$ are the lengths of shortest paths from (1) to i; $i = 1, 2, \ldots, N$. If the algorithm does not converge in N iterations (i.e. $\pi_i^{(N-1)} \neq \pi_i^{(N)}$, $i = 1, 2, \ldots, N$) it is because there exists at least a shortest path from (1) to some (i) that has more than N-1 arcs, i.e. a negative cycle.

The proof of the algorithm is based on the idea of breaking a shortest path into homogeneous blocks of nodes in which the numbers naming the nodes in each block form either a strictly increasing or decreasing sequence.

The situation can be depicted as follows for the shortest path from (1) to some (i).

$$\underbrace{(1) < (N_1) < (N_2) < \ldots < (N_{r_1})}_{\text{1st homogeneous block}} > \underbrace{(N_{r_1+1}) > (N_{r_1+2}) > \ldots}_{\text{2nd homogeneous block}}$$

$$\ldots > (N_{r_2}) < \ldots \underbrace{(N_{r_{M-1}+1}) \ldots (i)}_{\text{Mth homogeneous block}}$$

where $1 \leq M \leq N-1$

For example, the path

$$(1) - (4) - (5) - (2) - (11) - (9) - (8) - (12) - (13)$$

consists of 5 homogeneous blocks

$$|(1) < (4) < (5)| > (2)| < (11)| > (9) > (8)| < (12) < (13)|$$

In the first homogeneous block, since $(1) < (N_1) < (N_2) < \ldots < (N_{r_1})$, the optimal lengths from (1) to (N_1), (N_2), ..., (N_{r_1}) are determined in iteration 1, i.e. $\pi_i^{(1)}$. The optimal lengths from (1) to nodes in the second block are obtained in the second iteration. Continuing, the optimal lengths to nodes in the Mth block are obtained in the Mth iteration. The number M of homogeneous blocks is bounded above by N-1 since in the worst case (with N even) the shortest path from node (1) to node (N) could be as follows:

$$(1) \rightarrow (N-1) \rightarrow (2) \rightarrow (N-2) \rightarrow (3) \rightarrow (N-3) \ldots \rightarrow (k) \rightarrow (N)$$

where $k = N-k$

which contains N-1 homogeneous blocks of node numbers forming either a strictly increasing or decreasing sequence. Therefore, if there are no negative cycles in the network the

successive approximation of the shortest lengths from (1) to (i), i = 1, 2, ..., N terminates in no more than N-1 iterations. A flowchart of the algorithm is presented in Fig. 2-4.

A Modification to Yen's Algorithm

Yen's algorithm can be modified to reduce the amount of computational effort required to solve the shortest path problems. The computational savings are realized by restricting the minimization operation to a smaller subset of the functional equations than was required in the original algorithm. Recall that the recursive equations can be separated into even and odd iterations. The minimization during an odd (even) iteration is now performed only over those nodes which both have been previously processed and whose functional equation value has improved since the last odd (even) iteration.

This restriction is possible since each homogeneous block must contain at least one node and therefore at least one additional π_i becomes the correct minimum distance at each iteration and consequently no longer affects the calculations.

Two additional terminating criterion are introduced. Since the algorithm attempts to find the shortest paths from a source node to all nodes, should the functional equation corresponding to the source node become negative, this would indicate a path of negative length from the source node to itself or a negative cycle. This does not imply that

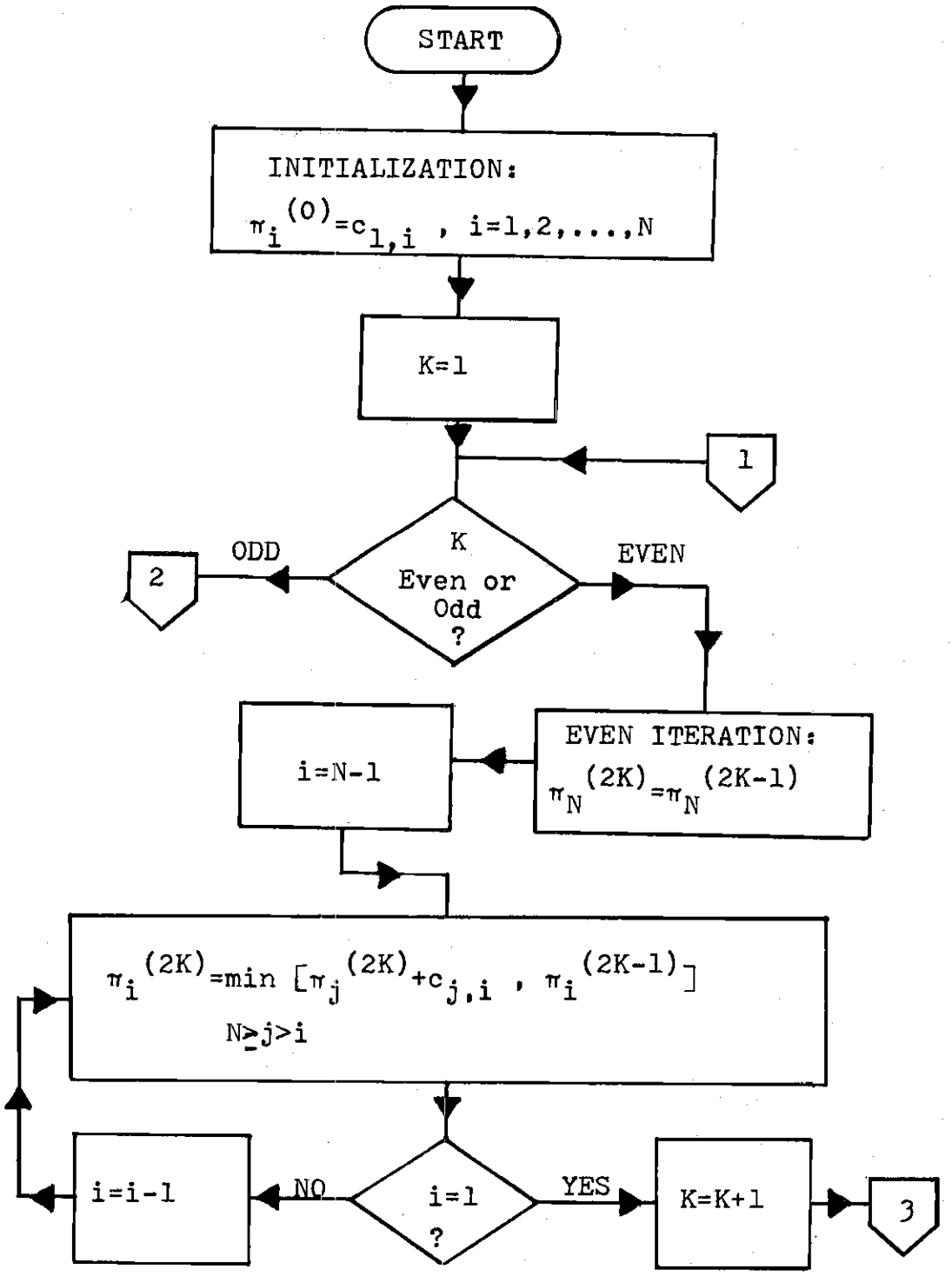


Figure 2-4. Flowchart of the Yen Algorithm

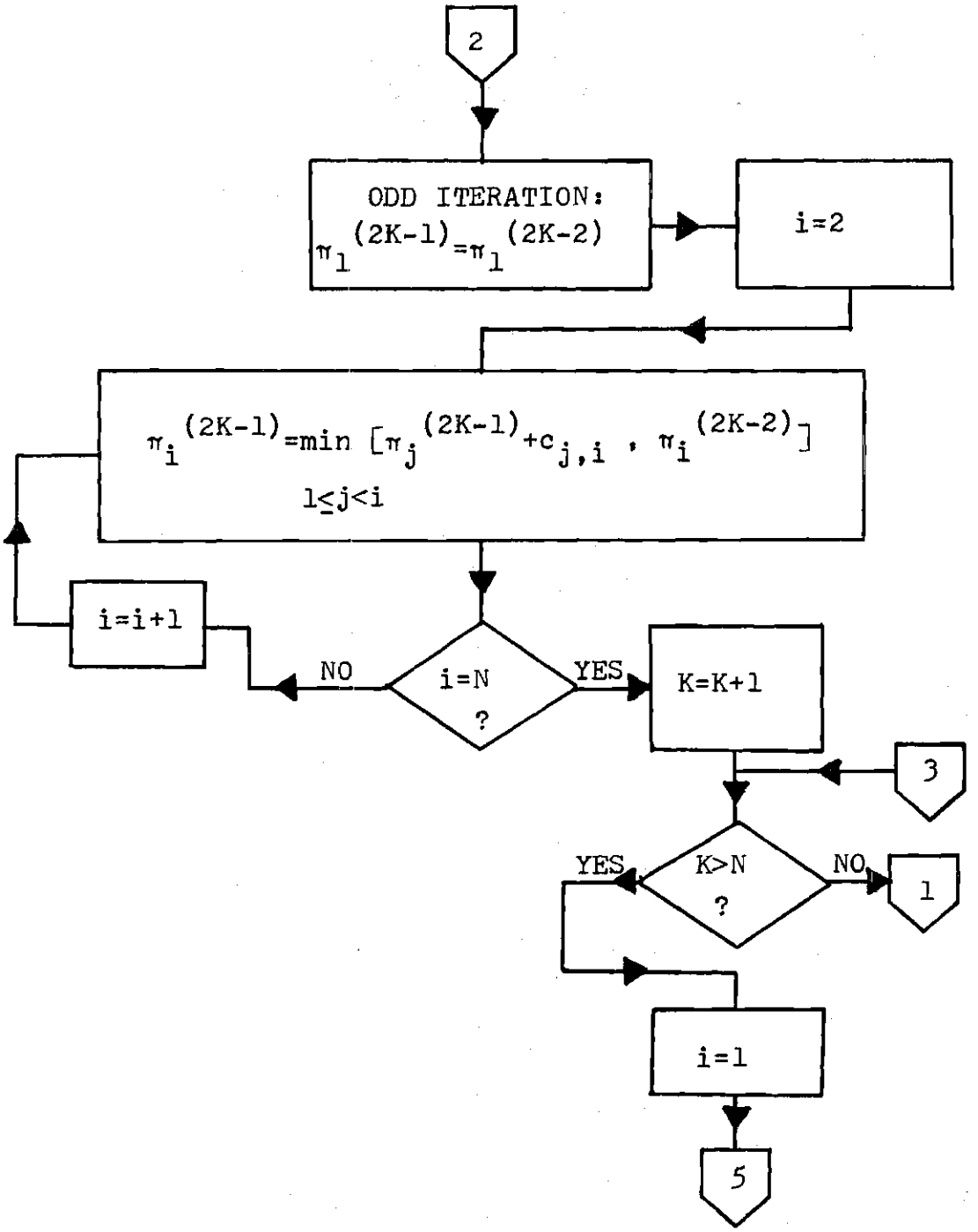


Figure 2-4 (continued)

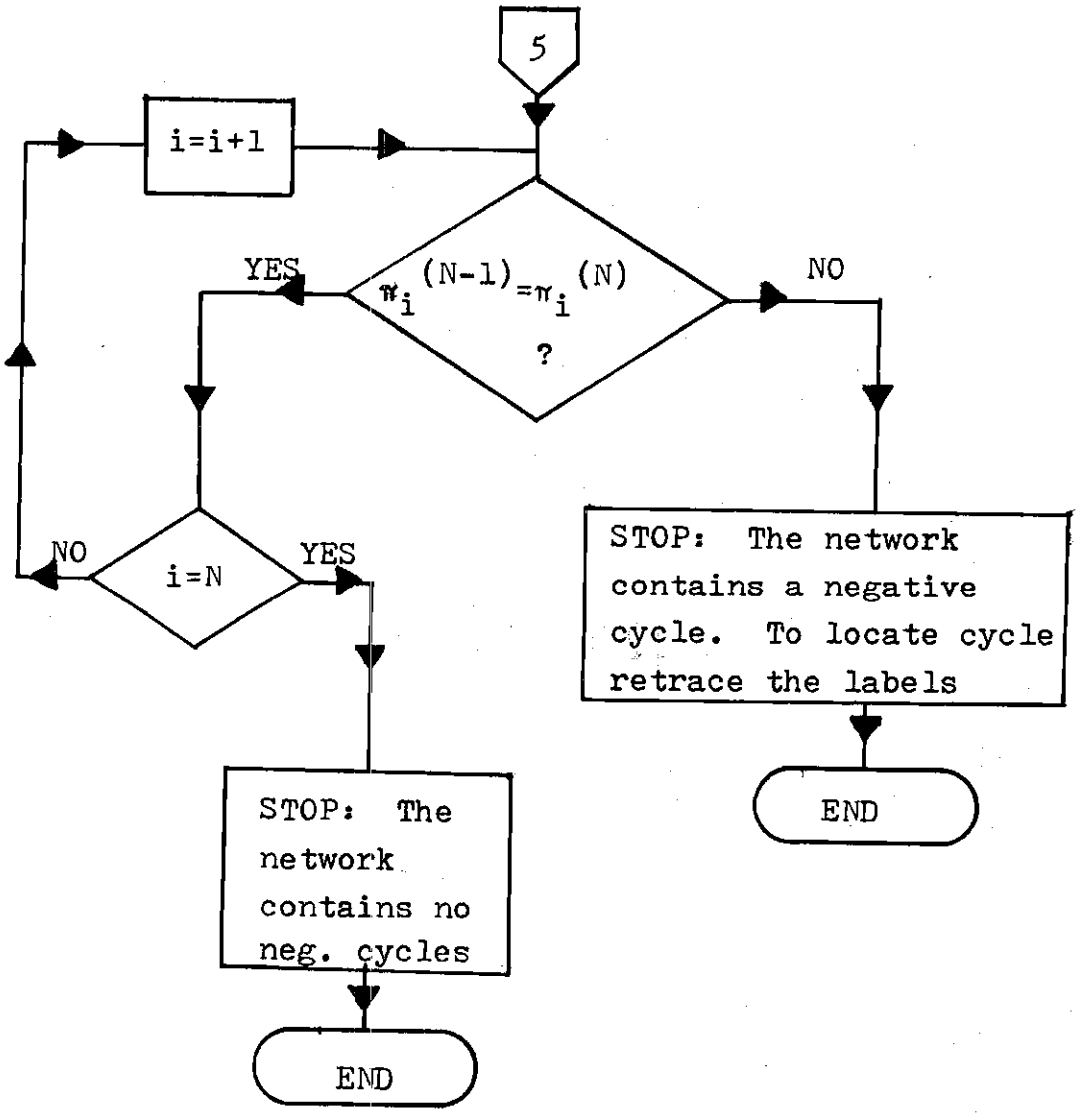


Figure 2-4 (concluded)

the source node is contained in a simple negative cycle as the cycle may occur elsewhere in the network as shown in Fig. 2-5. Also, if at the end of the kth iteration, which is an odd (even) iteration, the number of functional equations which have remained equal to their values in the last odd (even) iteration is less than k-2, then the network contains a negative cycle. This implies that starting with the M = 3rd iteration and continuing successively, one less functional equation is allowed to change its value over its value 2 iterations ago, than could have at the M-1st iteration. That is, during iteration 3, at most N-1 functional equations may change their value over their value in iteration 1. In iteration 4, at most N-2 functional equations may change their value over their value in iteration 2, etc.

The reasoning behind this terminating criterion is similar to that offered earlier for restricting the minimization operator to a subset of the previously processed nodes. That is, each homogeneous block must contain at least one node and therefore, at least one additional π_i becomes the correct minimum distance at each iteration and consequently no longer affects the calculations.

The third terminating criterion is similar to that proposed in the original algorithm. When $\pi_i^{(2k)} = \pi_i^{(2k-1)}$ or $\pi_i^{(2k-1)} = \pi_i^{(2k)}$, $i = 1, 2, \ldots, N$, the algorithm can be terminated. That is, when there is no change in the values of the functional equations between two successive iterations

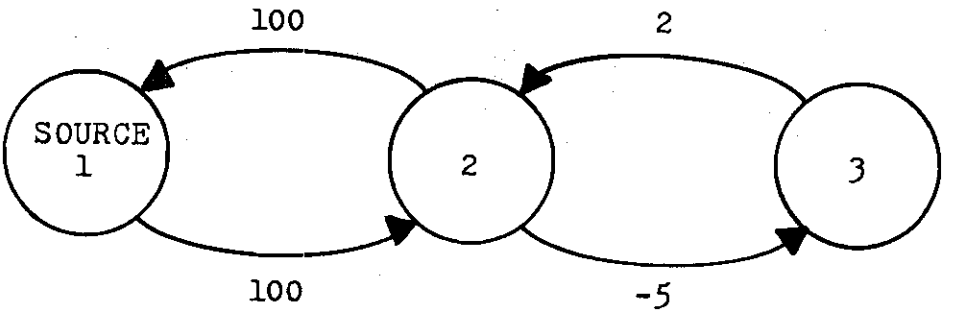


Figure 2-5. Source Node in a Non-Simple Negative Cycle

(j) and (j+1), then $\pi_i^{(j+1)}$, i = 1, 2, ..., N are the lengths of the shortest distances from node (1) to node (i). The new algorithm is expected to reduce the computational effort of the original algorithm by as much as 50%. An example of this procedure is presented in the appendix.

The Simplex Algorithm of Bennington

The linear programming formulation of the shortest path problem can be written as:

$$\text{minimize } z = \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij}x_{ij}$$

subject to:

$$\sum_{j=1}^{N} (x_{ij} - x_{ji}) = \begin{cases} 1 & i = 1 \\ 0 & i = 1,N \\ -1 & i = N \end{cases}$$

$$x_{ij} \geq 0$$

As noted earlier, Dantzig offered the first linear programming based algorithm to solve the shortest path problem. Bennington refined this work with his simplex algorithm to determine the shortest paths from an origin to all remaining nodes. Bennington proved that the simplex algorithm could be restricted to a subset of the basic feasible solutions corresponding to arborescences. An

arborescence centered at node s consists of a set of arcs $\mathcal{A}$ with the following properties:

1. $\mathcal{A}$ contains N-1 arcs
2. $\mathcal{A}$ contains no cycles
3. $\mathcal{A}$ contains a unique path from node s to node j for all $j \neq s$.
4. $\mathcal{A}$ contains no arcs (i,s) into node s and exactly one arc (i,j) into each node $j \neq s$.

Although there are basic feasible solutions that are not arborescences, given any basic feasible solution there is an equivalent arborescence, i.e. an arborescence which yields the same values for all of the primal variables, x_{ij}.

Choosing the origin s as node 1, the algorithm selects an arbitrary arborescence $\mathcal{A}$ centered at node 1. This arborescence corresponds to a trial (basic feasible) solution to the shortest path problem. The algorithm then computes the simplex multipliers by setting $\pi_1 = 0$ and using the recursive equation $\pi_i + c_{ij} = \pi_j$ for $(i,j) \in \mathcal{A}$. These simplex multipliers are equivalent to the negative of the dual variables in the dual linear programming problem. Next, the algorithm attempts to improve on the trial solution by pivoting into the basis an arc (p,q) such that $\pi_p + c_{pq} < \pi_q$. If no such arc exists, i.e. $\pi_i + c_{ij} \geq \pi_j$ for all (i,j), then $\mathcal{A}$ corresponds to the optimal solution and π_j are the lengths of the shortest paths from node 1 to node j for $j = 1, 2, \ldots, N$.

If an arc (p,q) satisfying $\pi_p + c_{pq} < \pi_q$ is found and if the path in $\mathcal{A}$ from node 1 to node p includes node q then the arc (p,q) along with the path from node q to node p in $\mathcal{A}$ will form a negative cycle. The proof is as follows: Let the present feasible solution be as in Fig. 2-6.

Since nodes p, q and s are in the arborescence, $\pi_p = \pi_s + c_{sp}$ and $\pi_s = \pi_q + c_{qs}$. If $\pi_p + c_{pq} < \pi_q$, we may bring the arc (p,q) into the basis. Upon substituting for π_p we have

$$(\pi_s + c_{sp}) + c_{pq} < \pi_q$$

Substituting for π_s

$$(\pi_q + c_{qs}) + c_{sp} + c_{pq} < \pi_q$$

Subtracting π_q from both sides

$$c_{qs} + c_{sp} + c_{pq} < 0$$

If the test for negative cycles fails, a new improved basis $\mathcal{A}'$ can be formed by deleting the arc $(i,q) \in \mathcal{A}$ and adding the arc (p,q). The simplex multipliers are recomputed and the algorithm continues until the shortest paths are found or a negative cycle is detected.

A flowchart of the algorithm is presented in Fig. 2-7.

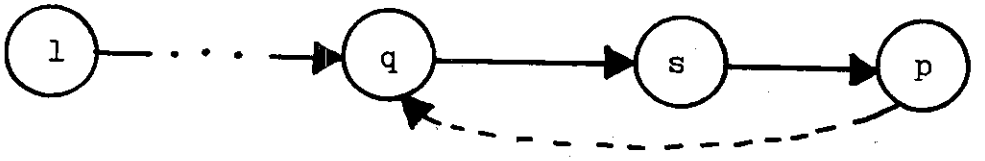


Figure 2-6. A Basic Feasible Solution in the Bennington Algorithm

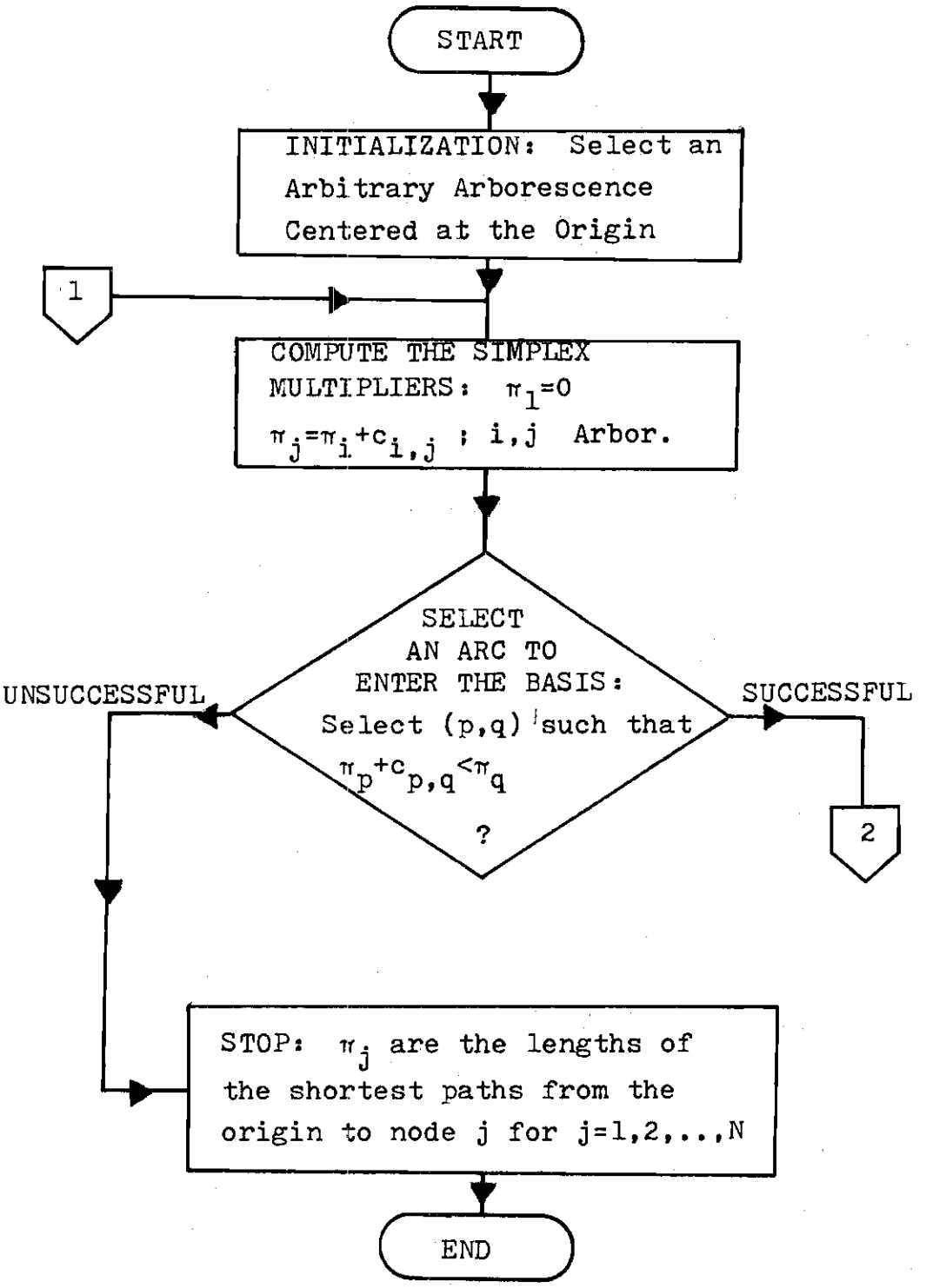


Figure 2-7. Flowchart of the Bennington Algorithm

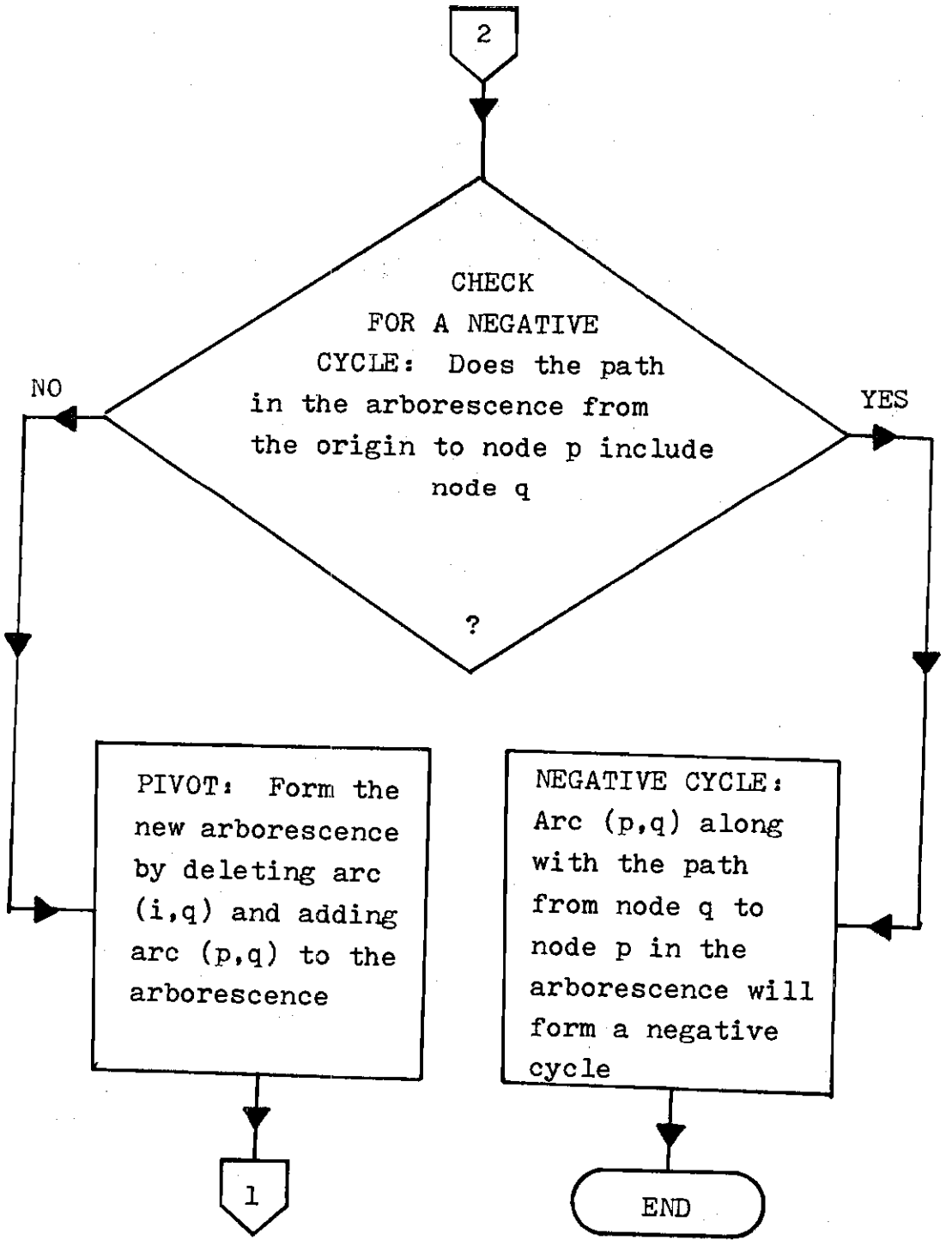


Figure 2-7 (concluded)

An example of the procedure is presented in the appendix.

The algorithm is particularly suited to the minimal cost flow problem since information used in detecting the kth negative cycle can be used in initiating the search for the k + 1st negative cycle. This distinguishes the simplex algorithm from the remaining algorithms which retain no information from previous iterations.

After locating a negative cycle and making the corresponding flow change around the cycle, we wish to construct the new marginal cost network and check it for negative cycles. Since the only arcs in the marginal cost network that are affected by the flow change are those contained in the negative cycle, an arborescence for the new marginal cost network can be easily obtained from the current solution. Modify the arborescence from the previous iteration by deleting the arcs in the path from node q to node p. Add all of the reverse arcs from the negative cycle except the reverse arc directed into node q. This forms an arborescence in the new marginal cost network. In summary then, Bennington's algorithm applied to the minimal cost flow problem proceeds as follows:

1. Construct the initial arborescence (centered at node N) in the marginal cost network. The reason for choosing node N will be explained subsequently.

2. Search the marginal cost network for a negative cycle by pivoting into the basis, appropriate non-basic

arcs (p,q) and tracing the path from node 1 to node p.

3. If a negative cycle is found, change the flows around the cycle.

4. Modify the arborescence as previously described.

5. Use the modified arborescence as an initial basic solution to search for negative cycles in the new marginal cost network.

Continue in this manner until a marginal cost network is created which is absent of negative cycles.

Therefore, although considerable time may be required to create the initial arborescence and locate the first negative cycle, over the course of solving a minimal cost flow problem which may require hundreds of iterations, the efficiency of Bennington's algorithm in utilizing information from previous iterations may enable it to outperform the others.

A difficulty which arises when Bennington's algorithm is applied to the minimal cost flow problem occurs when the source node is used to center the arborescence and the value of the flow is a maximum. In this case the Bennington algorithm may not be able to find every negative cycle. The reasoning is as follows: Let X and $\bar{X}$ be the sets of labeled and unlabeled nodes from the maximum flow algorithm. All of the arcs between X and $\bar{X}$ will be directed from $\bar{X}$ to X in the marginal cost network. Thus, there will be no directed paths in the marginal cost network from nodes in X to nodes

in $\bar{X}$ and a negative cycle in $\bar{X}$ cannot be detected. This difficulty can be overcome by centering the arborescence around the sink node and attempting to find the shortest path from the sink to all remaining nodes.

Direct Search Methods

There is only one available algorithm which is primarily concerned with locating negative cycles in a graph. The algorithm does not search for paths between pairs of nodes and therefore cannot determine the shortest paths in the situation where negative cycles do not exist.

The Method of Florian and Robert

Florian and Robert's Direct Search Method is based on a property of negative partial sums of finite sequences according to the following theorem:

Theorem III. A negative cycle exists in a network if, and only if, there exists a sequence of distinct nodes $i_1, i_2, \ldots, i_{r-1}$ with costs c_{ij} associated with the arcs (i,j) such that all partial sums of the sequence $c_{i_1 i_2}$, $c_{i_2 i_3}, \ldots, c_{i_{r-1} i_1}$ are negative. An example of the direct search method of characterizing a negative cycle is shown in Fig. 2-8.

The negative cycle passing through nodes 1, 2, and 3 can be written as:

(a) $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$

(b) $2 \rightarrow 3 \rightarrow 1 \rightarrow 2$

(c) $3 \rightarrow 1 \rightarrow 2 \rightarrow 3$

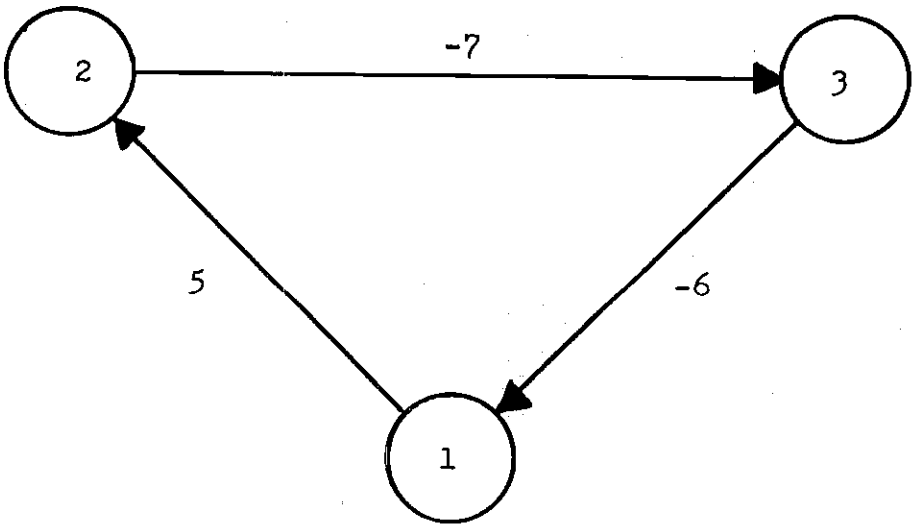


Figure 2-8. A Negative Cycle

Only representations "b" and "c" would satisfy Florian and Robert's property for identifying negative cycles since the partial sum of the sequence (1,2) in "a" is not negative. Note that this property does not uniquely classify a negative cycle as there may be more than one sequence which satisfies Theorem III, e.g. "b" and "c".

The result of this theorem suggests an algorithm for locating negative cycles in a graph. If a graph contains a negative cycle then there exists a node in the cycle such that the partial sums of arc lengths along the arc progression that starts and terminates at this node are all negative. The task of locating a negative cycle reduces to that of finding a node that has this property.

The algorithm starts at a home base node (node 1) and searches for a sequence of negative partial sums eminating from this node. If the negative sum of arc lengths cannot be preserved, this node is abandoned and another node is considered in the same way. The procedure continues until either a negative cycle is found or until all of the nodes have been examined for the negative partial sum property and discarded. A flowchart of the algorithm is presented in Fig. 2-9. An example of the procedure is presented in the appendix.

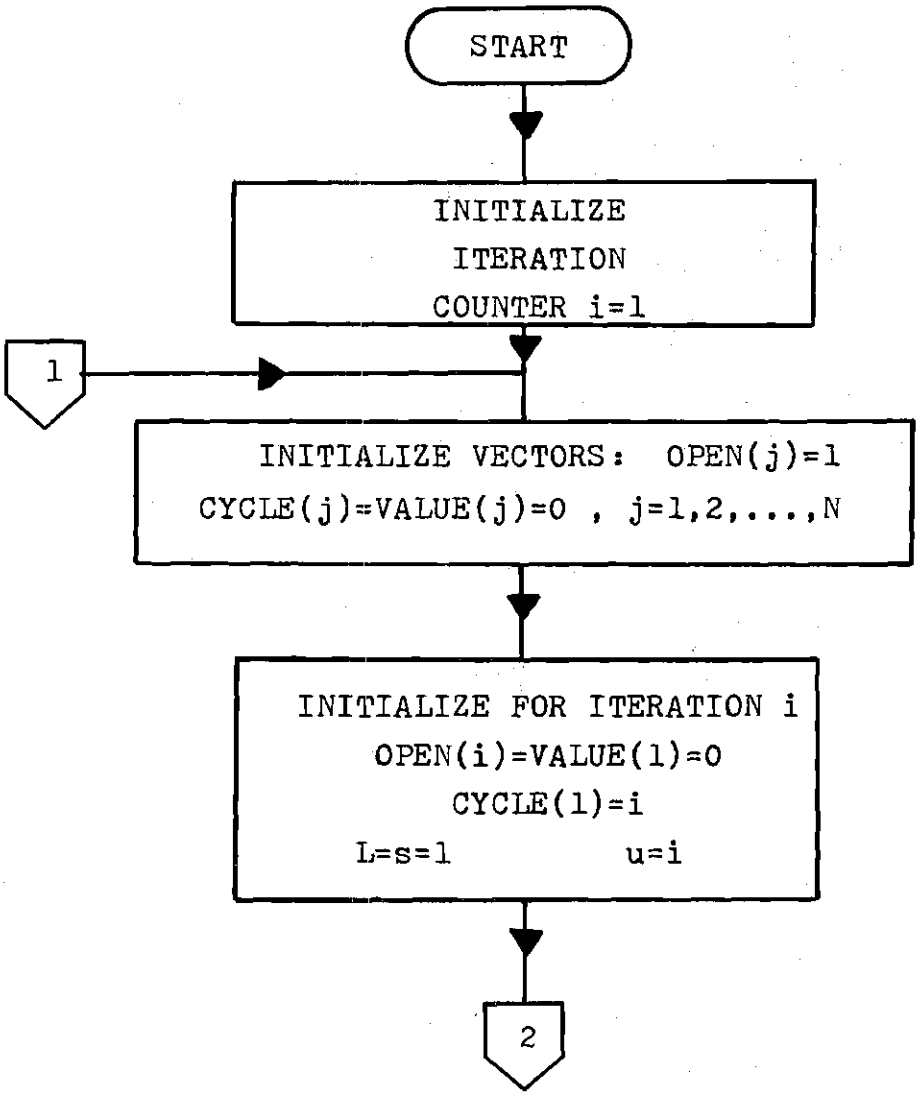


Figure 2-9. Flowchart of the Florian and Robert Algorithm

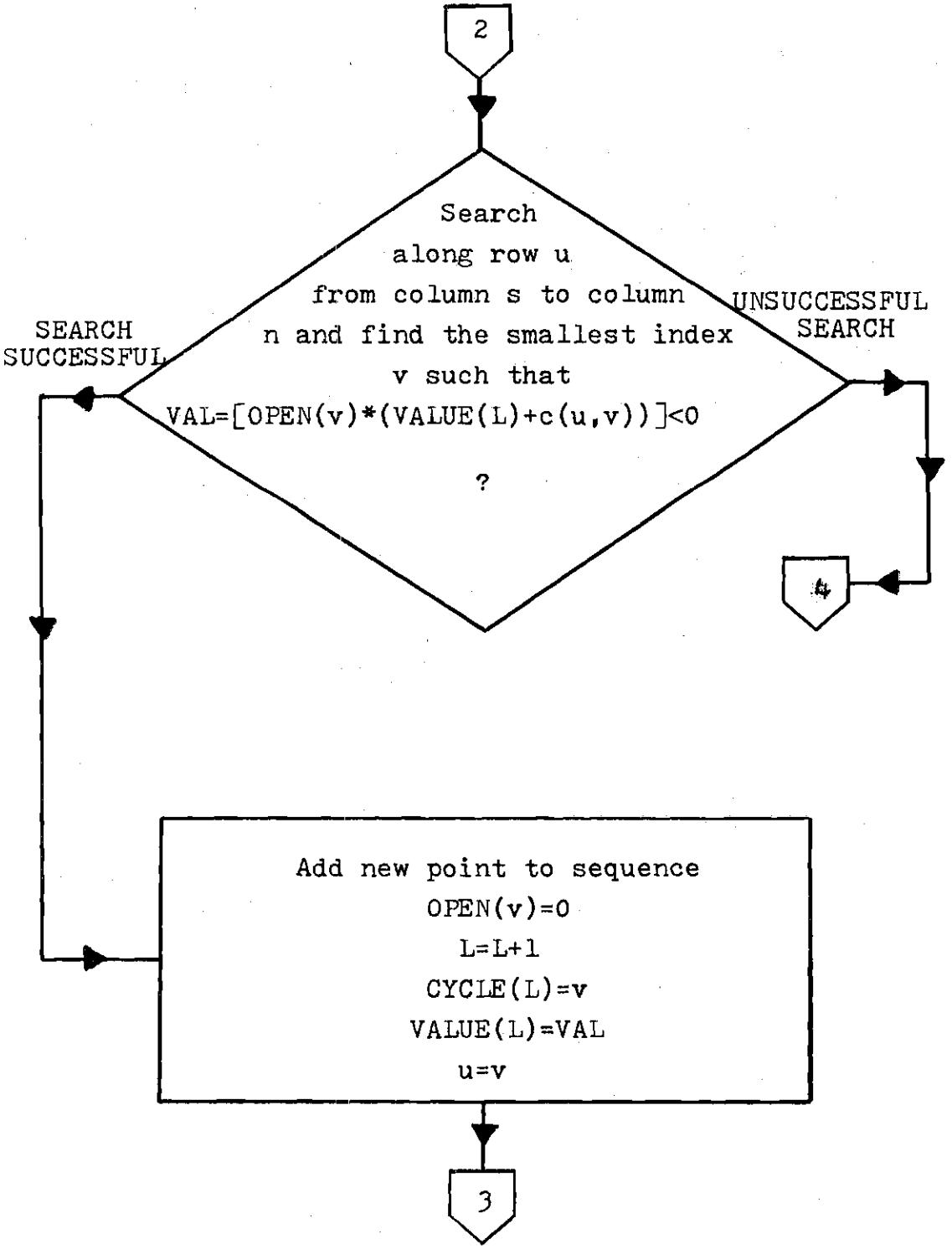


Figure 2-9 (continued)

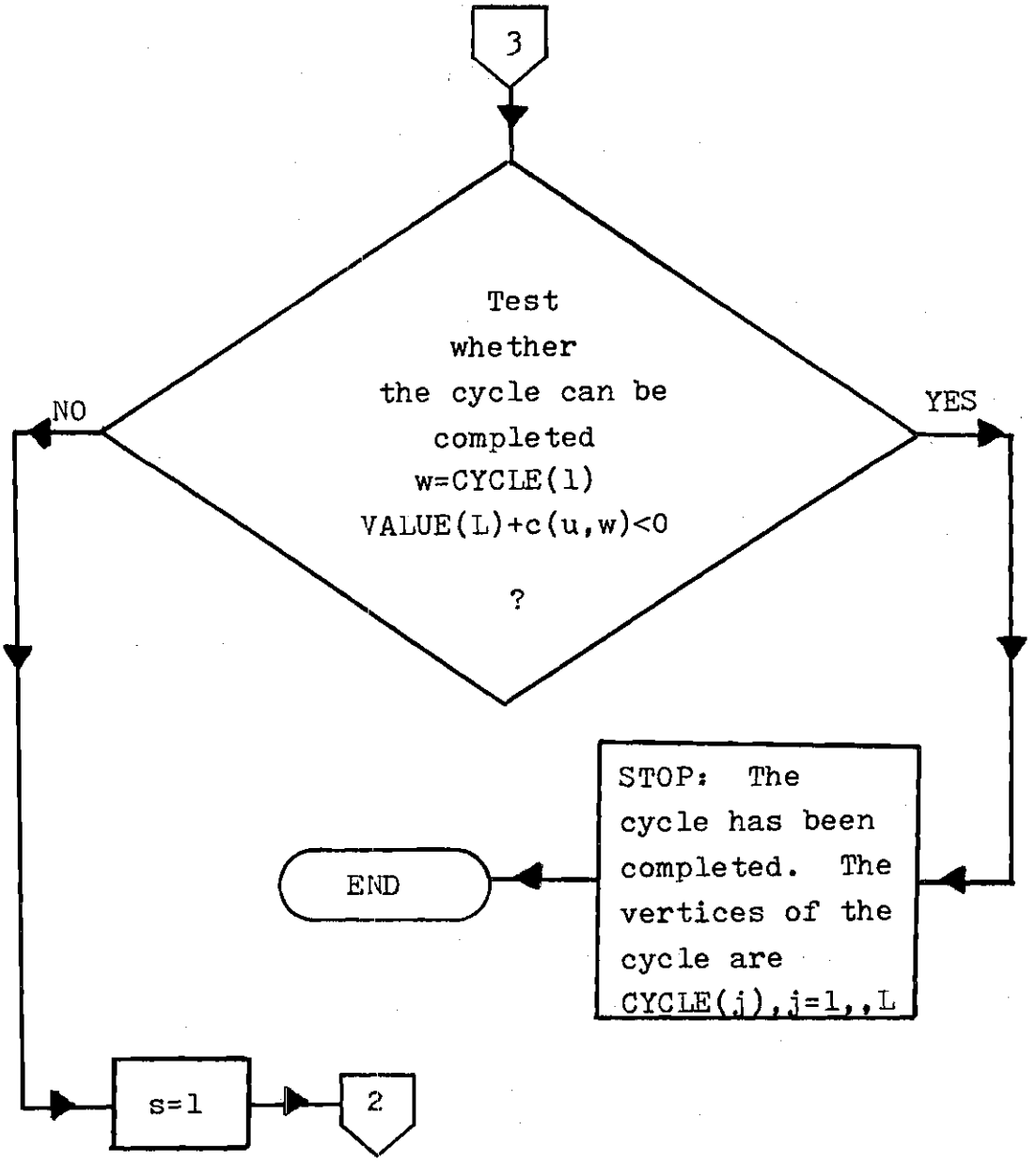


Figure 2-9 (continued)

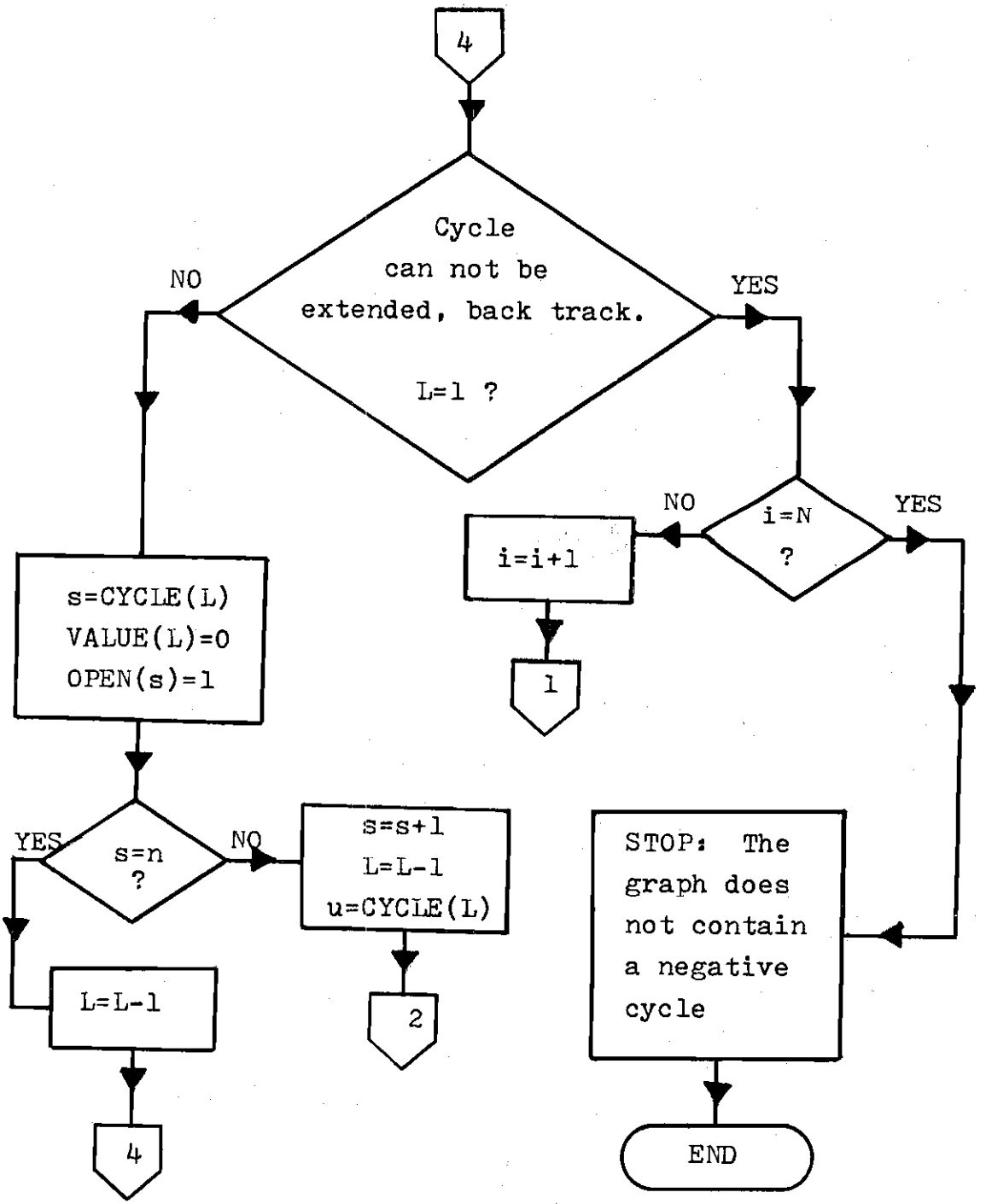


Figure 2-9 (concluded)

CHAPTER III

THEORETICAL UPPER BOUNDS

There are two commonly used methods of comparing the efficiency of algorithms. The first is an empirical measurement based on observing the performance of the algorithms over a range of problems. For network problems this range may be quite large as previous researchers have reported a number of independent factors which affect how quickly an algorithm converges. Among these are the number of nodes, the density of the network, along with the mean, variance, and shape of the distribution of arc distances. It may be impracticable to test the algorithms on every combination of the above factors. Therefore, a measure of efficiency based on empirical results can be misleading. For one type of problem an algorithm can be very efficient while for another type of problem it can be very inefficient.

A more conservative measurement of the efficiency of an algorithm is the upper bound on the number of computations required to reach an optimal solution. The computational upper bound specifies the maximum price users have to pay in the worst case; hence, the lower the computational upper bound, the "better" the algorithm in the worst case.

Therefore, as an initial critique we computed the

computational upper bound for each algorithm. These upper bounds can be considered as the maximum number of computations required to locate a single negative cycle assuming (1) a completely dense network and (2) no prior information on the network structure.

Ford-Fulkerson Algorithm

An iteration of the Ford-Fulkerson algorithm consists of using the label on nodes i, i = 1, 2, ..., N in sequence along with the cost associated with arc (i,j) to improve the label on node j, j = 1, 2, ..., N; $i \neq j$. This involves an addition, e.g. $\pi_i + c_{ij}$, and comparison, e.g. $\pi_i + c_{ij} < \pi_j$ for each i,j. Therefore, an iteration requires approximately N^2 additions and N^2 comparisons. Totaled over N iterations the algorithm requires approximately N^3 additions and N^3 comparisons to find the lengths of all shortest paths from a fixed node.

Yen Algorithm

In the Yen algorithm, iterations are classified as even or odd. As originally presented, nodes are processed in the decreasing sequence, N, N-1, ..., 1 during even iterations and in the increasing sequence 1, 2, ..., N during odd iterations. Minimizing over only those nodes previously processed necessitates half as many additions and comparisons per iteration than does the Ford-Fulkerson algorithm. The number of comparisons required per iteration

as a function of the processed node is presented in Table 3-1. An identical number of additions is required.

Table 3-1. Number of Comparisons Required in Yen's Algorithm

	Processed Node						
Iteration	1	2	3	...	N-2	N-1	N
Even	N-1	N-2	N-3	...	2	1	0
Odd	0	1	2	...	N-3	N-2	N-1

The sum of either sequence whether even or odd is $N(N-1)/2$. Summed over N iterations, the algorithm requires at most $N^2(N-1)/2$ additions and comparisons.

Florian and Robert's Algorithm

In theory, Florian and Robert's direct search method achieves its computational upper bound on problems of the following structure: (i) c_{ij}'s (the arc distances from node i to node j) are negative for all $i = 1, 2, \ldots, N$ and $j = 2, 3, \ldots, N$, $j \neq i$; (ii) c_{ij}'s are positive for $i = 2, 3, \ldots, N$ and $j = 1$; and (iii) the network has only one N-arc negative cycle which passes through nodes, 1, N, N-1, ..., 3, 2, and 1 in sequence.

To determine the existence of the negative cycles by taking node 1 as the home node, Florian and Robert's direct search method has to compute the lengths of all possible cycles around node 1 that contain 2 arcs, 3 arcs, ..., N

arcs before it can locate the N-arc negative cycle. There are

$$\binom{N-1}{2-1}(2-1)!, \binom{N-1}{3-1}(3-1)!, \ldots, \binom{N-1}{N-1}(N-1)!$$

ways of forming cycles with 2 arcs, 3 arcs, ..., N arcs, respectively. Therefore, the upper bound on the number of cycles which the direct search method must compute the length of, to solve the problem is

$$\sum_{i=1}^{N-1} \binom{N-1}{i} \cdot i!$$

Computing the length of a cycle using the direct search method requires at least an addition and (the same number of) comparisons. Therefore, on problems of the above structure, the direct search method requires at least

$$\sum_{i=1}^{N-1} \binom{N-1}{i} \cdot i!$$

additions and comparisons to detect the negative cycle.

As an illustration, consider a network with the following distance matrix:

Let C_{ij} =

	1	2	3	4
1	0	-1	-2	-3
2	25	0	-2	-4
3	30	-1	0	-2
4	30	-2	-22	0

To determine the existence of the negative cycles by taking node 1 as the home node, the direct search method has to compute the lengths of all cycles around node 1 that contain 2 arcs, i.e. [1-2-1, 1-3-1, 1-4-1], 3 arcs, i.e. [1-2-3-1, 1-2-4-1, 1-3-2-1, 1-3-4-1, 1-4-2-1, 1-4-3-1], 4 arcs, i.e. [1-2-3-4-1, 1-2-4-3-1, 1-3-2-4-1, 1-3-4-2-1, 1-4-2-3-1] before it can determine the existence of our particular 4-arc negative cycle 1-4-3-2-1.

The direct search method has to compute the lengths of

$$\sum_{i=1}^{3} \binom{3}{i} \cdot i! = 15$$

different cycles to solve the problem.

It is unlikely that many "real-life" problems will possess the above structure. What is possible though, are problems in which a long sequence of negative partial sums is created with no possible path returning to the home base node. The above bound has shown that it may be relatively dangerous to apply the direct search method to networks possessing even a few of the attributes of the above problem.

Bennington's Algorithm

The Bennington algorithm is the most difficult for which to calculate an upper bound. Computing the bound becomes complicated since it is dependent on the manner in which the arborescence is created and stored. A summary of the steps in the algorithm and a reasonable bound on each step is as follows:

1. Create the initial arborescence--From the origin node scan the remaining (N-1) nodes to determine which nodes can be connected to the origin. If no arcs are available, connect a single node to the origin via an artificial arc. From the second node in the arborescence scan the remaining nodes (at most N-2) to determine which can be connected to this node. Continue in this manner until the arborescence is completed. Therefore, at most N-1 + N-2 + ... 1 = N(N-1)/2 comparisons are required to construct the arborescence.

2. Compute the simplex multipliers--As a node i is placed in the arborescence, its corresponding simplex multiplier π_i is computed. A single addition is required to compute each multiplier. Therefore, over the entire network, N-1 additions are required, i.e. π_1 is initially set to zero.

3. Select an arc to enter the basis--Select an arc (p,q) such that $\pi_p + c_{pq} < \pi_q$. There are N*(N-1) arcs in the network and only N-1 arcs are in the present basis.

Consequently, there are N*(N-1) - (N-1) nonbasic arcs to scan. Checking each arc requires an addition and comparison, hence, at most N*(N-1) - (N-1) additions and comparisons are required. Since there are N*(N-1) arcs in the network and checking each arc requires an addition and comparison, at most N*(N-1) additions and comparisons are required.

4. Check for a negative cycle--This involves tracing the path from node 1 to node p to determine if it includes node q. Assuming no negative cycles, that is, the path will not include node q, the path could therefore contain N-2 arcs. Each encountered node along this path must be compared to nodes 1 and q. Therefore, at most 2*(N-2) comparisons are required.

5. Pivot and recompute the simplex multipliers--Assuming the arcs in the arborescence are stored in a list, to recompute the multipliers requires N-1 additions.

Note that Step 2 is identical to Step 5. Step 2 is performed during the initial iteration and Step 5 is performed during all remaining iterations. If we substitute Step 5 for Step 2, the final three steps (Steps 3, 4, 5) are repeated at most (N-1)! times corresponding to the (N-1)! possible arborescences. Therefore, the total number of operations required is less than

$$\frac{N*(N-1)}{2} + (N-1)! * [2*((N*(N-1)) - (N-1)) + 2*(N-2) + N-1]$$

It should be noted that this upper bound is rarely attained.

Summary

The present chapter can be summarized by stating the general bound on each algorithm and an estimate of how frequently the bound will be attained.

Ford-Fulkerson

The theoretical upper bound on the Ford-Fulkerson algorithm is of the order N^3. When using the algorithm to detect negative cycles, this bound is a good indication of the number of additions and comparisons required. The exception is when the source node is contained in a negative cycle (not necessarily a simple cycle, see Fig. 2-5). In this case, the algorithm will terminate in k, $2 \leq k \leq N$ iterations and will require kN^2 additions and comparisons.

Yen

The theoretical upper bound on the Yen algorithm is of the order $N^3/2$. The algorithm is similar to the Ford-Fulkerson algorithm and its upper bound will be realized to a similar extent.

Florian and Robert

The upper bound on the algorithm proposed by Florian and Robert is of exponential order. The bound will be attained on problems of the structure described earlier. The bound will be achieved to a lesser degree on problems containing only a few of the characteristics of the "worst case" example.

Bennington

The theoretical upper bound on the Bennington algorithm is also of an exponential order but is seldom attained. Experience with the simplex algorithm in general has shown that rarely is the maximum number of pivots required.

CHAPTER IV

EXPERIMENTAL DESIGN

An experiment was designed to study the effect of several factors on the detection of negative cycles in a graph. The literature was searched to determine those factors which previous researchers had reported most affected the computational efficiency of algorithms for network problems.

Golden [19] states that the performance of shortest-path algorithms is dependent upon the following three factors: (1) the sparseness of the network, (2) list processing and network representation in the computer code, and (3) distance measures on the arcs. Yen [31] reports that the mean, the variance and the shape of the distribution of arc distances have a very strong effect on how quickly an algorithm converges.

After analyzing the negative cycle problem we decided on four independent variables for the experiment. The factors consisted of (1) the specific algorithm used to locate the cycle, (2) the number of nodes in the network, (3) the density of the network, and (4) the distribution of arc costs.

The algorithm effect is obvious and is the primary

concern of this thesis.

The time required to complete a single iteration in each algorithm is a function of the number of nodes in the network, i.e. the greater the number of nodes which must be examined, the greater the number of additions and comparisons involved.

The density of the network is defined as the ratio of the number of arcs in the network to the number of possible arcs if all pairs of nodes were connected, i.e. (#arcs/N(N-1)). Sparse networks will contain fewer arcs and hence should require less computational effort than more dense problems. A dense network though, may contain a number of negative cycles. These cycles correspond to alternate optimal solutions and therefore a dense network may converge quickly.

When the arc costs are generated from a uniform distribution centered at zero (u=0), the distribution can be uniquely characterized by a single parameter, the variance. Intuitively, it would seem that the variance would influence the value of the negative cycle, i.e. the sum of the costs associated with the arcs around the cycle. When the uniform distribution is centered at zero, a large variance will result in several arcs having a "large negative" cost. The shortest path algorithms essentially search for the "least cost" arc at each node and therefore should locate cycles with relatively large negative values.

The arc distribution factor may influence the

computational time of the Florian and Robert algorithm. A negative cycle consisting of k arcs, in a network whose arcs are distributed with a large variance, may contain one "large negative" arc and k-1 positive arcs. Therefore, if an arc out of the home base node has an associated "large negative" cost, it may be possible to construct a negative cycle around the home base utilizing this single negative arc.

A list processing approach was employed so the effect of network representation would be common to all four algorithms. Under list processing, only those arcs which exist in the network are stored in computer memory. Since the networks are sparce, this approach will tend to accelerate the algorithms. Three lists are used to store the originating nodes i, the terminating nodes j, and the cost associated with arc (i,j) respectively. Two additional lists serve as pointers to indicate arcs entering or leaving a desired node.

A classical factorial design consisting of four factors, algorithm, nodes, density, and arc distribution was selected with each factor evaluated at (4,4,4,3) levels, respectively. Networks were generated and searched for negative cycles using all possible combinations of the following factors:

A. Type of algorithm--the algorithm used to detect and trace negative cycles (Yen, Bennington, Florian-Robert, Ford-Fulkerson).

B. Nodes--the number of nodes in the network (25,50,75,100).

C. Density--the ratio of the number of arcs in the network to the number of possible arcs if all pairs of nodes were connected (.05,.10,.15,.20).

D. Distribution of arc costs--the costs associated with the arcs were generated from a uniform distribution with mean = 0, over three ranges corresponding to a required variance. The intervals chosen were (-25,25), (-125,125), and (-250,250) corresponding to a variance of 208, 5208, and 20833, respectively.

The problem size was restricted to 100 nodes due to limitations in computer storage. The levels of density and arc distribution were selected to reflect previous work in the field. The specific levels of the three quantitative factors were chosen at the extremes and at intermediate levels so as to cover the entire range of interest. By choosing more than two levels for each factor, we can detect both cubic and quadratic effects on the dependent or response variables.

After some thought on expected variability under the same set of conditions, it was decided to take three observations under each of the 192 conditions, making a total of 576 runs.

A mathematical model for this experimental design

would be:

$$Y_{ijkmn} = u + A_i + B_j + C_k + D_m + AB_{ij} + AD_{im}$$

$$+ BC_{jk} + BD_{jm} + CD_{km} + \varepsilon_{n(ijkm)},$$

where Y_{ijkmn} represents the measured variable, u a common effect in all observations (the true mean of the population from which the data came), A_i the algorithm effect where $i = 1,2,3,4$, B_j the node effect where $j = 1,2,3,4$, C_k the density effect where $k = 1,2,3,4$ and D_m the distribution of arc costs where $m = 1,2,3$. $\varepsilon_{n(ijkm)}$ represents the random error in the experiment where $n = 1,2,3$. The other terms stand for interactions between the main factors A,B,C,D. The higher order (three way and above) interactions were pooled with the error term, since they would be difficult to explain if found significant.

Model Selection

After carefully weighing the ramifications, a fixed effects model was chosen for the experiment. The decision was difficult since the random effects model possesses some advantages over a fixed effects model. Specifically, the random effects model enables the experimentor to generalize statistically beyond the M levels actually observed. This contrasts the fixed effects model in which the results are valid only for those levels examined.

The choice of model would not affect the analysis of algorithms since our interest in this factor is confined to the four algorithms. Further, the primary concern of the thesis is in comparing algorithms. Therefore, since the algorithm effect is a fixed effect, our choice of model will not influence the decision on the performance of any algorithm.

The advantages of the random (mixed) effects model would be realized when analyzing the remaining factors (nodes, density, and arc distribution) for which we have no interest in any one particular level. For example, it would be beneficial to extend any inferences drawn on the sample of nodes in the experiment to the population of all nodes. Unfortunately, there is a high price associated with the random effects model. This model assumes that the levels to be tested have been chosen randomly from a very large population of potential groups. A random choice of nodes though, might not cover the range of interest. A compromising situation arises. The information gained by being able to extend the results to all levels might be lost through a poor sample of levels in the experiment.

An additional difficulty which arises when choosing the type of model to employ in a design, concerns replicating the experiment. Replication will enable the experimenter to gain a more precise estimate of the error variance. To replicate a fixed-effects experiment, we would obtain more observations from the M levels used in the original experiment.

To replicate a random-effects experiment we would have to pick a new sample of M levels from which to take observations. Our concern over the sample of levels covering the range of interest is intensified when replications are desired. That is, if the probability of a sample covering the range of interest is p, then the probability of three successive samples covering the range of interest is $p^3 < p$. Thus, again a fixed effects model is preferred over a random effects model when replications are appropriate in a quantitative factor such as nodes.

In our experiment, replications and the range considerations were considered to be more important than the inferences to be drawn, so the random effects model was rejected in favor of the fixed effects model. For a more detailed explanation of the two models, the reader is referred to [11].

Selection of Response Variables

The initial response variable considered was the computational time required to locate and trace a negative cycle in a graph. Since each algorithm could not be expected to locate the identical negative cycle, an additional response variable of a quality index was also recorded. The quality index was defined as the ratio of the absolute value of the sum of the arc costs around the cycle to the time required to locate and trace the cycle. For example, if the negative cycle in Fig. 4-1 required 2 sec to locate and trace

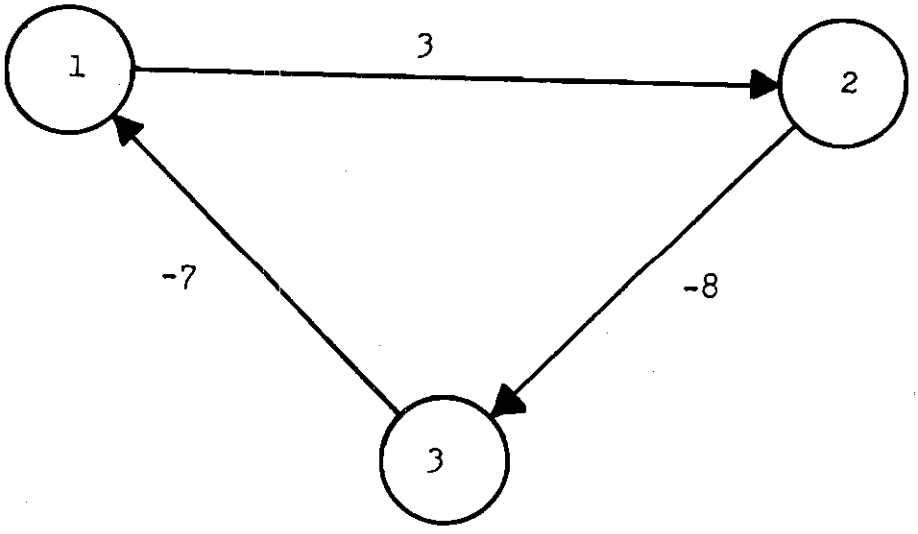


Figure 4-1. A Negative Cycle

it would have an associated quality index of

$$\frac{|3-8-7|}{2} = \frac{|-12|}{2} = 6$$

The higher the quality index, the better the performance of the algorithm based on the supposition that cycles of large negative value will enable Klein's algorithm for the minimum cost flow problem to converge in few iterations.

Additional response variables of interest include the value of the cycle (the sum of arc costs around the cycle) and the number of nodes in the negative cycle.

Generating Random Networks

The generation of networks possessing various combinations of the foregoing factors was an integral element of the experimental design. In order to obtain a valid comparison of the four algorithms, it was essential to test them under identical conditions. That is, a network would be generated with a specified number of nodes, density, and distribution of arc costs and each algorithm would receive this network as input and proceed to search for negative cycles.

To guarantee a connected graph, i.e. every pair of nodes are joined by at least one chain, the first N-1 arcs were generated to form an arborescence in the network. Recall that an arborescence centered at node s, is a directed path containing no cycles, from node s to all remaining nodes.

Upon completion of the arborescence, additional arcs were generated to complete the required density.

The following procedure was employed to create the random networks:

Generating Random Networks Containing an Arborescence

Let X be the set of nodes in the arborescence. Originally X = {∅}. Choose a node s at random to center the arborescence. Let $\bar{X}$ be the set of nodes not yet in the arborescence. Originally $\bar{X}$ = {1, 2, ..., N}.

NOTE: $X \cap \bar{X} = \{\emptyset\}$

$X \cup \bar{X} = \{1, 2, \ldots, N\}$

STEP 1 Remove node s from $\bar{X}$ and place it in X

STEP 2 Randomly select a node j in $\bar{X}$. Place node j in X and arc (s,j) in the network. Remove node j from $\bar{X}$.

STEP 3 Randomly select a node i in X. Randomly select a remaining node j in $\bar{X}$. Remove j from $\bar{X}$ and place it in X. Place arc (i,j) in the network.

STEP 4 If $\bar{X} = \{\emptyset\}$, the arborescence centered at node s is now complete. Otherwise, go to STEP 3.

Let NREQ be the number of arcs required for a specified density d.

$$NREQ = d*N*(N-1)$$

Since an arborescence contains N-1 arcs we must add NREQ - (N-1) additional arcs to the network to obtain the required

density. The procedure used to generate these additional arcs is listed in the appendix.

Generating an Initial Feasible Solution for the Simplex Algorithm

The previous discussion of the Bennington algorithm in Chapter II assumed that an initial basic feasible solution was at hand. This initial solution corresponds to an arborescence centered at the origin (node 1 in our experiment). Our initial arborescence used to obtain a connected graph is centered around a randomly chosen node. Therefore, we cannot guarantee the existence of an arborescence centered at node 1.

We can resolve this problem by creating an artificial arborescence around node 1. This procedure would be similar to the Big-M method in linear programming.

Creating an Artificial Arborescence

All nodes are initially unlabeled and unscanned. The origin (node 1) is labeled [-,0]. These labels are similar to those used in the Ford-Fulkerson shortest path algorithm (Chapter II). The first element of the ordered pair corresponds to the preceding node in a path from the origin to the labeled node. The second element is the distance along this path or the sum of the arc costs on the path.

We begin by labeling all nodes j which can be reached from the origin using one arc (1,j). These nodes receive the label [1, u(j) = c(1,j)] and are considered as labeled

and unscanned while the origin is now labeled and scanned. If there are no arcs leaving the origin, an artificial arc is used to connect the origin to node 2. Node 2 would be considered labeled [1, u(2) = ∞] and unscanned while the origin is labeled and scanned. Next a labeled but unscanned node j is chosen. Using this node j, attempt to label any unlabeled nodes k which can be reached from node j in one arc (j,k). These nodes k would be considered as labeled [j, u(k) = u(j) + c(j,k)] and unscanned while node j is now labeled and scanned. Continue in this manner using labeled and unscanned nodes to label unlabeled nodes. When all nodes have been scanned, artificial arcs are used to connect any unlabeled nodes m to the origin. These nodes are labeled with [1, u(m) = ∞].

We now have an arborescence (possibly artificial) centered at node 1 to initiate the simplex method. These artificial arcs will have an associated cost of ∞, but cannot be pivoted out of the basis since they were added only as a last resort when no legitimate arc existed. Therefore, should this initial arborescence contain any artificial arcs, and should the algorithm converge and find the set of shortest paths from the origin (thereby implying the absence of negative cycles) this set of shortest paths must be considered infeasible.

CHAPTER V

EMPIRICAL RESULTS

The experiment was performed by generating 144 networks (corresponding to three replications of the factorial design) and applying each of the four algorithms to locate a negative cycle.

Initially, a number s was randomly generated on the interval (1,N) where N is the number of required nodes. The number s corresponded to the node which would center the initial arborescence. After the N-1 arcs were generated to construct the arborescence, [(d*N*N-1) - (N-1)] additional arcs were generated to complete the required density (where d=density).

The following response variables were recorded throughout the experiment: (1) computational time to detect and trace the negative cycle, (2) value of the cycle, (3) quality of the negative cycle, i.e. |(2)|/(1), (4) the number of nodes in the cycle, and (5) the distinct nodes of the cycle. A summary of the quality indices and solution times is presented in Tables 5-1 and 5-2.

An analysis of variance (Table 5-3) performed on the quality index data revealed a significant effect (at the 0.05 level) due to algorithm, arc distribution, and the algorithm-

Table 5-1. Mean Response--Quality Index

Algorithm		Nodes	
Yen	7189.8	25 Nodes	12945.8
Bennington	1517.5	50 Nodes	12965.6
Florian-Robert	42137.0	75 Nodes	13322.1
Ford-Fulkerson	880.0	100 Nodes	12491.0

Density		Arc Distribution	
.05	12529.6	(-25,25)	2923.4
.10	14000.7	(-125,125)	12231.8
.15	13563.7	(-250,250)	23638.0
.20	11630.5		

Table 5-2. Mean Response--Computational Time (Sec)

Algorithm		Nodes	
Yen	0.085	25 Nodes	0.101
Bennington	0.809	50 Nodes	1.134
Florian-Robert	0.873	75 Nodes	0.548
Ford-Fulkerson	1.277	100 Nodes	1.260

Density		Arc Distribution	
.05	1.335	(-25,25)	0.607
.10	0.601	(-125,125)	0.563
.15	0.498	(-250,250)	1.112
.20	0.609		

Table 5-3. Analysis of Variance--Quality

Source	Degrees of Freedom	Sum of Squares (1)	Mean Square (1)	F-ratio
Main Effects				
Algorithm	3	167.00	55.60	191.72*
Nodes	3	0.05	0.02	0.07
Density	3	0.49	0.16	0.55
Arc Distribution	2	41.30	20.60	71.03*
Interactions				
Algorithm-Nodes	9	1.25	0.14	0.48
Algorithm-Density	9	3.01	0.33	1.14
Algorithm-Arc Distribution	6	76.40	12.70	43.79*
Nodes-Density	9	3.54	0.39	1.34
Nodes-Arc Distribution	6	2.04	0.34	1.17
Density-Arc Distribution	6	0.41	0.07	0.24
Error	519	151.00	0.29	

*Significant at $\alpha = 0.05$ level

(1) Entries are in 10^9

Table 5-4. Analysis of Variance--Computational Time

Source	Degrees of Freedom	Sum of Squares	Mean Square	F-ratio
Main Effects				
Algorithm	3	106.30	35.40	1.56
Nodes	3	125.00	41.67	1.84
Density	3	64.30	21.43	0.95
Arc Distribution	2	35.70	17.85	0.79
Interactions				
Algorithm-Nodes	9	426.68	47.40	2.09*
Algorithm-Density	9	281.18	31.24	1.38
Algorithm-Arc Distribution	6	127.13	21.18	0.93
Nodes-Density	9	318.37	35.37	1.56
Nodes-Arc Distribution	6	115.59	19.26	0.85
Density-Arc Distribution	6	112.36	18.72	0.83
Error	519	11729.70	22.60	

*Significant at $\alpha = 0.05$ level

arc distribution interaction. A similar analysis of variance executed on computational time (Table 5-4) disclosed an algorithm-node interaction as the single significant effect. This algorithm-node interaction reveals that as the number of nodes in the network is increased, its effect on computational time differs among the algorithms chosen to locate the cycle. The sum of squares due to all nonsignificant effects were pooled together with the sum of squares due to error and the analysis was reperformed. The results of the ANOVA conformed with the original analysis.

We can utilize the mean response data in explaining these effects. Table 5-1 shows that on the average, negative cycles obtained with the algorithm of Florian and Robert have an associated quality almost six times larger than any other algorithm. There is no discernible difference in quality over the levels of nodes or density. The mean response over arc distribution shows a perceptible increase in quality over successive levels of this variable.

The results in Table 5-2 question the superiority of the Florian and Robert algorithm. The mean computational time for this algorithm is exceeded only by the Ford-Fulkerson algorithm. Since quality index is inversely proportional to computational time, these figures challenge the validity of the experiment.

In addition to the contrasting results over quality and computational time with the Florian and Robert algorithm,

the lack of any significant effect due to nodes is also reason for concern. Since an iteration in each algorithm is a function of the number of nodes, we had originally expected an increase in the computational time (and therefore a decrease in quality index) with an increase in the number of nodes.

Examination of Table 5-5 which contains the mean responses over all combinations of algorithms and nodes offers some insight into the problem. The dependency on nodes is verified by the responses from three of the four algorithms. The algorithm of Florian and Robert is unaffected by the number of nodes in the network with the exception of the mean computational time for 50 node networks which is exceptionally high. This term is responsible for the significant algorithm-node interaction uncovered by the ANOVA in Table 5-4.

Inspection of the results over the individual networks provides the rationale for the bizarre results. In our experiment the direct search method located a cycle on each of the 144 networks in 125.69 sec; 142 of the networks were solved with an average computational time of 0.0045 sec, compared to the overall average time of 0.873 sec. The remaining two networks required 11.59 sec and 113.45 sec, respectively, for solution. Both problems occurred on low (.05) density 50 node networks. This latter term accounts for the high algorithm-node term for computational time in

Table 5-5. Mean Response--Algorithm-Node Interaction

	Nodes	Quality	Computational Time (sec)
Yen	25	5423.8	.062
	50	8627.2	.089
	75	7674.4	.078
	100	7033.8	.108
Bennington	25	4153.1	.056
	50	1274.2	.325
	75	410.2	.798
	100	232.4	2.058
Florian-Robert	25	40150.7	.0050
	50	41141.5	3.4760
	75	44801.4	.0046
	100	42454.5	.0054
Ford-Fulkerson	25	2055.3	.2820
	50	819.3	.6440
	75	402.5	1.3130
	100	243.1	2.8680

Table 5-5. Recall from Chapter III the theoretical upper bound on the number of required computations for Florian's direct search method. On these 50 node problems, the direct search method could perform as many as

$$\sum_{i=1}^{49} \binom{49}{i} i\ !$$

additions and comparisons before locating a negative cycle. When computing the theoretical upper bound for this algorithm there was no evidence to suggest a dependency on density or level of nodes. The occurrence of the upper bound on low density 50 node problems would seem to transpire strictly by chance.

Estimating the Effects in the Mathematical Model

Recall from Chapter IV that the mathematical model is:

$$Y_{ijkmn} = u + A_i + B_j + C_k + D_m + AB_{ij} + AC_{ik} + AD_{im}$$

$$+ BC_{jk} + BD_{jm} + CD_{km} + \varepsilon_{n(ijkm)}$$

In the analysis of variance we are interested in testing hypotheses of the form:

$$H_0 : \alpha_i = 0, \quad i = 1, 2, \ldots, a$$

$$H_1 : \alpha_i = 0, \quad \text{at least one } i$$

or:

$$H_0 : (\alpha\beta)_{ij} = 0, \quad i = 1, 2, \ldots, a \qquad j = 1, 2, \ldots, b$$

$$H_1 : (\alpha\beta)_{ij} = 0, \quad \text{at least one } i \text{ and } j$$

The first null hypothesis states that the treatments denoted by A have no effect on the response, while the second null hypothesis states that there is no interaction between A and B that influences the response. For those calculated values of the F-statistic from Tables 5-3 and 5-4 which exceed a predetermined (critical) value of the F-statistic we can reject the associated null hypothesis and accept the alternative hypothesis that the effect in question does indeed influence the response. For these effects we may obtain an estimate of their magnitude in our mathematical model by the method of least squares [21]. In addition we may compute confidence intervals for the true value of the parameters [10]. These confidence intervals will enable us to compare the estimates and determine which levels of the significant factors are statistically different. Estimates of the effects and 95% confidence intervals for the estimates are presented in Table 5-6 through Table 5-7.

We decided to accept only a 5% chance of rejecting a null hypothesis when it is true. Therefore, we conclude that the Florian and Robert algorithm is the only algorithm

Table 5-6. Estimate of Effects

Quality

$\hat{u}$ = 12931

Main Effects

Algorithm	Estimate
Yen	-5741.75
Bennington	-11413.75
Florian-Robert	29206.25
Ford-Fulkerson	-12050.75
Confidence Interval	±15036.30

Arc Distribution	Estimate
(-25,25)	-10007.30
(-125,125)	-698.95
(-250,250)	10707.25
Confidence Interval	±12277.10

Table 5-7. Estimate of Effects

Interactions

	Arc Distribution		
Algorithm	(-25,25)	(-125,125)	(-250,250)
Yen	4389.0	1848.0	-6235.0
Bennington	8760.0	892.0	-9651.0
Florian-Robert	-22442.0	-3222.0	25663.0
Ford-Fulkerson	9293.0	485.0	-9779.0
Confidence Interval	±21264.6	±21264.6	±21264.6

Computational Time

$\hat{u} = 0.760$

Interactions

	Nodes			
Algorithm	25	50	75	100
Yen	0.637	-0.365	0.206	-0.476
Bennington	-0.094	-0.854	0.202	0.742
Florian-Robert	-0.207	2.229	-0.654	-1.365
Ford-Fulkerson	-0.335	-1.002	0.246	1.092
Confidence Interval	±5.680	±5.680	±5.680	±5.680

whose affect on the quality of negative cycles is significantly different from zero. The effects of the remaining algorithms all lie within a 95% confidence interval centered at zero, and hence, we cannot distinguish between them. The positive estimate for the Florian and Robert algorithm indicates that this algorithm is superior in locating negative cycles of high quality.

The estimates of the effect due to arc distribution are significantly different only between the low and high levels. Examining the estimates for the algorithm-arc distribution interaction effect and their associated confidence intervals indicates that the Florian and Robert algorithm has a significant effect at both the low and high level of arc distribution. The algorithm performs poorly on networks with small variability in the arc costs but is the superior of the four algorithms on networks whose arcs are distributed with a large variance. The negative estimate for the Florian-Robert algorithm under networks whose arcs are distributed with a small variance is a result of the two network problems which approached the upper bound. The estimates of the algorithm-node interaction effect on computational time all lie within a 95% confidence interval. The estimate for the effect of Florian and Robert's algorithm on 50 node networks is considerably higher than the remaining algorithms but does not fall outside the confidence interval.

We cannot have a large degree of confidence in the

foregoing results since we do not have a homogeneous system. The results on the two network problems which approached the upper bound in the algorithm of Florian and Robert should be considered as "missing values" since they do not provide a rational measure of performance for this algorithm. The occurrence of networks containing the characteristics described in Chapter III is a random phenomenon and cannot be associated with any particular level of the factors. Therefore, the outcomes from these two networks have distorted the analysis. Repeating the experiment might also give rise to networks which approach the theoretical upper bound. Consequently, a more appropriate analysis would be conducted over the three remaining algorithms.

Analysis--Excluding Florian and Robert's Direct Search Method

The algorithm of Florian and Robert was removed and an analysis of variance performed on the remaining data. The mean response data is presented in Tables 5-8 and 5-9 while the analyses are shown in Tables 5-10 and 5-11.

The analysis on quality produced a significant effect due to all main factors: algorithm, nodes, density, and arc distribution. In addition, all interactions involving algorithms were found to be significant. The main effects can be verified by examining the mean responses in Table 5-8. The Yen algorithm produces negative cycles with the highest mean quality. The difference in mean quality between the Yen and Ford-Fulkerson algorithms seems large enough to create an

Table 5-8. Mean Response--Quality-Excluding Florian and Robert's Algorithm

Algorithm		Nodes	
Yen	7189.8	25	3877.4
Bennington	1517.5	50	3573.6
Ford-Fulkerson	880.0	75	2829.1
		100	2503.1

Density		Arc Distribution	
.05	2013.0	(-25,25)	668.7
.10	3249.6	(-125,125)	3570.8
.15	3733.7	(-250,250)	5347.9
.20	3786.9		

Table 5-9. Mean Response--Computational Time-Excluding Florian and Robert's Algorithm

Algorithm	
Yen	0.085
Bennington	0.809
Ford-Fulkerson	1.277

Nodes	
25	0.133
50	0.353
75	0.730
100	1.678

Density	
.05	0.621
.10	0.800
.15	0.662
.20	0.811

Arc Distribution	
(-25,25)	0.728
(-125,125)	0.750
(-250,250)	0.694

Table 5-10. Analysis of Variance--Quality-Excluding Florian and Robert's Algorithm

Source	Degrees of Freedom	Sum of Squares (1)	Mean Square (1)	F-ratio
Main Effects				
Algorithm	2	34.70	17.30	113.80*
Nodes	3	1.32	0.44	2.89*
Density	3	2.20	0.73	4.82*
Arc Distribution	2	16.00	8.00	52.60*
Interactions				
Algorithm-Nodes	6	4.97	0.83	5.44*
Algorithm-Density	6	5.40	0.90	5.92*
Algorithm-Arc Distribution	4	11.40	2.85	18.70*
Nodes-Density	9	1.92	0.21	1.40
Nodes-Arc Distribution	6	0.68	0.11	0.74
Density-Arc Distribution	6	0.85	0.14	0.92
Error	384	58.49	0.15	

*Significant at $\alpha = 0.05$ level

(1) Entries are in 10^8

Table 5-11. Analysis of Variance--Computational Time-Excluding Florian and Robert's Algorithm

Source	Degrees of Freedom	Sum of Squares	Mean Square	F-ratio
Main Effects				
Algorithm	2	103.93	51.96	108.50*
Nodes	3	150.88	50.30	105.07*
Density	3	3.00	1.00	2.09
Arc Distribution	2	0.23	0.12	0.24
Interactions				
Algorithm-Nodes	6	75.19	12.53	26.17*
Algorithm-Density	6	16.84	2.81	5.87*
Algorithm-Arc Distribution	4	0.29	0.07	0.14
Nodes-Density	9	14.21	1.58	3.30*
Nodes-Arc Distribution	6	5.56	0.93	1.94
Density-Arc Distribution	6	1.80	0.30	0.63
Error	384	183.82	0.48	

*Significant at $\alpha = 0.05$ level

effect due to algorithm. This data shows an increasing complexity of network problems with an increase in nodes. Increasing density supplies the networks with additional arcs and leads to negative cycles of high quality. The data also indicates that distributing the arc costs over wide intervals centered at 0 will result in negative cycles with high quality.

The analysis on computational time produced a significant effect due to algorithms and nodes. The algorithm-nodes, algorithm-density and nodes-density interactions were also considered to be significant. Table 5-9 substantiates the effects due to algorithm and nodes. Surprisingly, there was no effect on computational time due to density. Networks with high densities contain a relatively large number of arcs and would seem to be more complex than low density networks. Obviously, the analysis has demonstrated otherwise. The lack of an arc distribution effect is understandable since the variance of arc costs would not seem to be a computational factor.

Before estimating the significant effects the sum of squares due to all nonsignificant effects were pooled together with the sum of squares due to error and the analysis was reperformed. The results of the ANOVA conformed with the original analysis.

Estimating the Effects in the Mathematical Model

To determine those levels of effects which were statistically dissimilar, estimates were computed for the magnitude of each significant effect using the method of least squares. A 95% confidence interval was calculated for the true value of each effect. The results are shown in Tables 5-12 through 5-15.

The estimates of algorithm effect on quality are all significantly different from zero. These estimates show the Yen algorithm to be superior to either the Bennington or Ford-Fulkerson algorithms. The performance of the latter two algorithms does not deviate enough to distinguish between them. In the previous analysis the effects of these three algorithms were obscured due to the bizarre performance of the Florian and Robert algorithm.

The estimates for the node factor illustrate a decreasing quality with increasing nodes. These estimates though, are statistically significant only between 25 node and 75 or 100 node networks and between 50 node and 100 node networks.

The significant effect due to density is a result of low density (.05) networks. The quality of negative cycles located on these networks is much lower than the quality associated with any higher level of density tested. Within the remaining three levels of density there is no significant difference in quality.

The significance of the arc distribution results from

Table 5-12. Estimate of Effects--Excluding Florian and Robert

Quality

$\hat{u} = 3195.5$

Main Effects

Algorithm	Estimate
Yen	3993.00
Bennington	-1678.00
Ford-Fulkerson	-2315.00
Confidence Interval	± 743.13

Nodes	Estimate
25	681.50
50	377.50
75	-366.50
100	-692.50
Confidence Interval	± 910.15

Density	Estimate
0.05	-1182.20
0.10	53.70
0.15	537.70
0.20	590.70
Confidence Interval	± 910.15

Table 5-13. Estimate of Effects--Excluding Florian and Robert

Arc Distribution	Estimate
(-25,25)	-2527.00
(-125,125)	375.00
(-250,250)	2152.00
Confidence Interval	±743.13

Interactions

	Nodes			
Algorithm	25	50	75	100
Yen	-2448.00	1060.00	851.00	536.0
Bennington	1954.00	-621.00	-741.00	-593.00
Ford-Fulkerson	493.00	-439.00	-112.00	55.00
Confidence Interval	±1287.14	±1287.14	±1287.14	±1287.14

	Density			
Algorithm	0.05	0.10	0.15	0.20
Yen	-2301.00	-462.00	1011.00	1752.00
Bennington	1573.00	199.00	-723.00	-1051.00
Ford-Fulkerson	725.00	262.00	-289.00	-701.00
Confidence Interval	±1287.14	±1287.14	±1287.14	±1287.14

Table 5-14. Estimate of Effects--Excluding Florian and Robert

	Arc Distribution		
Algorithm	(-25,25)	(-125,125)	(-250,250)
Yen	-3092.00	773.00	2320.00
Bennington	1279.00	-183.00	-1096.00
Ford-Fulkerson	1812.00	-590.00	-1224.00
Confidence Interval	±1050.95	±1050.95	±1050.95

Computational Time

$\hat{u} = 0.724$

Main Effects

Algorithm	Estimate
Yen	-0.639
Bennington	0.086
Ford-Fulkerson	0.553
Confidence Interval	±0.234

Nodes	Estimate
25	-0.590
50	-0.370
75	0.006
100	0.954
Confidence Interval	±0.287

Table 5-15. Estimate of Effects--Excluding Florian and Robert

	Interactions			
	Nodes			
Algorithm	25	50	75	100
Yen	0.568	0.376	-0.012	0.930
Bennington	-0.163	-0.113	-0.018	0.294
Ford-Fulkerson	-0.405	-0.262	0.030	0.637
Confidence Interval	±0.405	±0.405	±0.405	±0.405

	Density			
Algorithm	0.05	0.10	0.15	0.20
Yen	0.188	-0.109	0.032	-0.111
Bennington	-0.023	0.372	-0.152	-0.199
Ford-Fulkerson	-0.174	-0.263	0.115	0.305
Confidence Interval	±0.405	±0.405	±0.405	±0.405

	Density			
Nodes	0.05	0.10	0.15	0.20
25	0.186	-0.020	-0.009	-0.155
50	0.339	-0.155	-0.071	-0.113
75	-0.104	-0.077	0.102	0.079
100	-0.425	0.246	-0.026	0.184
Confidence Interval	±0.497	±0.497	±0.497	±0.497

the low and high levels of this factor since these estimates are not contained in a 95% confidence interval centered at zero. These estimates display an increasing quality with increasing variance in the arc costs.

Examination of the estimates for algorithm-node interaction indicates that the Bennington algorithm is superior to the Ford-Fulkerson algorithm on 25 node networks which, in turn, is superior to the Yen algorithm on the identical networks. The estimates for the Bennington and Ford-Fulkerson algorithms are contained in the same confidence interval over the remaining three levels of nodes. The Yen algorithm is superior over the three remaining levels of nodes. At the high level (100 nodes) this superiority is not statistically significant.

The estimates for algorithm-density interaction indicate the Yen algorithm to be superior over all levels of density with the exception of 5% and 10% dense networks. On 5% dense networks, the Yen algorithm is inferior to either the Bennington or Ford-Fulkerson algorithms. At 10%, there is no statistically significant difference among the three algorithms. There is no detectable difference between the performance of the Bennington or Ford-Fulkerson algorithms over levels of density. The significant interaction is due to an increase in quality with increase in density for the Yen algorithm while the remaining two algorithms show a decrease in quality over the same range of densities.

The Yen algorithm is inferior to the Bennington or Ford-Fulkerson algorithms at the low level of arc distribution (-25,25). The opposite is true at the high level (-250,250). At the intermediate level, the performance of the Yen algorithm is superior only to the Ford-Fulkerson algorithm. There is no detectable difference between the Bennington or Ford-Fulkerson algorithms over any level of arc distribution. The increasing efficiency of the Yen algorithm over increasing levels of arc distribution along with a decreasing efficiency by the Bennington and Ford-Fulkerson algorithms led to the significant interaction.

Examining the estimates for computational time shows a considerable difference among all three algorithms. The Yen algorithm is faster than the Bennington algorithm, which in turn, is faster than the Ford-Fulkerson. However, only the Yen and Ford-Fulkerson algorithms have estimates which are significantly different from zero.

There is a discernible difference in computational time over each level of nodes with the exception of progressing from 25 nodes to 50 nodes. 50 node problems require a larger computational time than 25 node problems but both estimates fall within the same confidence interval. Only the estimate for 75 node problems does not significantly affect computational time.

The estimates for algorithm-node interaction indicate

the Yen algorithm to be inferior on 25 and 50 node networks. This algorithm is superior on 100 node networks. At 75 nodes, the performance of the three algorithms is essentially the same. There is no significant difference in computational time between the Bennington and Ford-Fulkerson algorithms over the range of node levels. The significant interaction is a result of the increasing efficiency of the Yen algorithm with increasing nodes. The remaining two algorithms become less effective as the number of nodes in a network is increased.

Examining the algorithm-density interaction effect on computational time shows a significant difference between the Yen and Bennington algorithms and between the Ford-Fulkerson and Bennington algorithms on 10% dense networks. The Yen and Ford-Fulkerson algorithms are superior to the Bennington algorithm at this level. There is also a considerable difference between the Yen and Ford-Fulkerson algorithms and between the Bennington and Ford-Fulkerson algorithms on 20% dense networks. The Yen and Bennington algorithms are superior to the Ford-Fulkerson algorithm at this level. The estimates of all remaining algorithm-density effects fall within the same confidence interval.

The node-density interaction was the least significant effect on computational time. The estimates of node-density effects confirm this, as discernible differences are evident only between the levels of nodes on 5% dense networks.

At this low level of density, 100 node problems decrease the computational effort required, whereas 25 and 50 node problems increase the effort.

Selecting the Most Effective Algorithm

It seems reasonable to conclude that on networks for which the theoretical upper bound is not attained, the Florian and Robert algorithm will produce the superior response in terms of both quality and computational time. Networks of the structure required to reach the upper bound occur at random and may not represent a real world problem. For the user who does not wish to gamble on the performance of this algorithm, we have constructed a chart based on estimates of the effects, displaying the most effective of the remaining three algorithms for a given set of network conditions. The algorithm which should produce negative cycles with the highest quality is given in Table 5-16. The algorithm which should locate a negative cycle in the shortest amount of time is given in Table 5-17. Note that the tables include only those factors which significantly affect the response.

For the user who desires a negative cycle with high quality, the Yen algorithm is favored under most conditions. The two exceptions are on 25 node networks at the low level of arc distribution (-25,25) and on networks both at the low level of arc distribution and low level of density (.05). On these networks the Bennington algorithm is recommended.

Table 5-16. Superior Algorithm on the Basis of Network Composition

Density/Arc Distribution		Quality: Nodes 25	50	75	100
.05	(-25,25)	2	2	2	2
.05	(-125,125)	2	1	1	1
.05	(-250,250)	1	1	1	1
.10	(-25,25)	2	1	1	1
.10	(-125,125)	1	1	1	1
.10	(250,250)	1	1	1	1
.15	(-25,25)	2	1	1	1
.15	(-125,125)	1	1	1	1
.15	(-250,250)	1	1	1	1
.20	(-25,25)	2	1	1	1
.20	(-125,125)	1	1	1	1
.20	(-250,250)	1	1	1	1

1 = Yen

2 = Bennington

3 = Ford-Fulkerson

Table 5-17. Superior Algorithm on the Basis of Network Composition

Computational Time

Density/Arc Distribution		Nodes 25	50	75	100
.05	all	2	1	1	1
.10	all	1	1	1	1
.15	all	2	1	1	1
.20	all	2	1	1	1

1 = Yen

2 = Bennington

3 = Ford-Fulkerson

For the user who desires the fastest computational time, the Yen algorithm is suggested on all networks with the exception of 25 node networks where the Bennington algorithm is preferred.

Analysis on the Value of the Cycle

The third response recorded on each network was the value of the negative cycle. Recall that the value of the negative cycle is defined as the sum of the costs associated with the arcs around the cycle. The data partitioned by algorithm, nodes, density, and arc distribution is shown in Table 5-18. Notice that the largest, in terms of "most negative," cycles correspond to the shortest path algorithms of Ford-Fulkerson and Yen. The direct search method does not concern itself with the value of the negative cycle and this is one of the disadvantages of the algorithm. As noted by Florian, [13], "The direct search method could be considerably improved, if a modification could be devised to insure that cycles of 'large' negative value are located in the first few iterations." Surprisingly, the Bennington algorithm, which is also a shortest-path algorithm, produced on the average, the smallest (least negative) cycle. A possible explanation lies in the selection of the non-basic arc to enter the basis. The simplex algorithm pseudo-arbitrarily selects an arc (p,q) (i.e. p is initially node 1 and proceeds toward node N following the sequence 1, 2,

Table 5-18. Mean Response--Value of the Cycle

Algorithm		Nodes	
Yen	-367.7	25 Nodes	-179.3
Bennington	-122.9	50 Nodes	-247.3
Florian-Robert	-200.3	75 Nodes	-325.6
Ford-Fulkerson	-447.6	100 Nodes	-386.3

Density		Arc Distribution	
.05	-230.7	(-25,25)	-53.9
.10	-263.7	(-125,125)	-278.7
.15	-309.0	(-250,250)	-521.3
.20	-335.0		

..., N) such that $\pi_p + c_{pq} < \pi_q$. If instead, when choosing a non-basic arc to enter the basis, we chose the most negative $\pi_p + c_{pq} - \pi_q$ we could expect the simplex algorithm to produce a "more negative" cycle. Since this would involve sorting arcs with an associated increase in computational time, the approach was dismissed as infeasible.

From Table 5-18, note the surprising increase (more negative) in the value of the negative cycle with an increase in the number of nodes in the network, an increase in the density, and an increase in the variance of the arc distribution. The associated analysis of variance in Table 5-19 confirms these claims. The significant effect due to algorithm was expected and is a result of the shortest path algorithms since shortest path algorithms, by searching for the "least cost" path, are in essence searching for the "most negative" cycle. The effect due to density can be explained since by providing a shortest path algorithm with a more dense network, you are offering it additional arcs with which to find "shorter" paths. The arc distribution effect as explained in Chapter IV is a result of providing those networks with a large variance, arcs with large negative costs. The shortest path algorithms will utilize these arcs to form cycles with large negative values. The effect due to nodes is similar to that of density in that it is the result of additional arcs in the network. A 25 node network with a density of .05 contains 30 arcs while a 50

Table 5-19. Analysis of Variance--Value of the Cycle

Source	Degrees of Freedom	Sum of Squares (1)	Mean Square (1)	F-ratio
Main Effects				
Algorithm	3	96.10	32.00	69.70*
Nodes	3	35.30	11.70	25.50*
Density	3	9.32	3.10	6.77*
Arc Distribution	2	210.00	105.00	228.70*
Interactions				
Algorithm-Nodes	9	32.10	3.57	7.77*
Algorithm-Density	9	23.80	2.64	5.75*
Algorithm-Arc Distribution	6	48.00	8.00	17.40*
Nodes-Density	9	6.65	0.74	1.60
Nodes-Arc Distribution	6	15.30	2.55	5.55*
Density-Arc Distribution	6	4.17	0.69	1.51
Error	519	238.00	0.46	

*Significant at $\alpha = 0.05$ level

(1) Entries are in 10^5

node network with the same density contains 122 arcs.

Analysis on the Number of Nodes in the Cycle

The mean number of nodes in a negative cycle partitioned by algorithm and number of nodes in the network is recorded in Table 5-20. These results show that, on the average, the direct search method contains the largest number of nodes in a negative cycle.

This outcome is surprising since the direct search method attempts to complete a cycle, by returning to the home base node (originally node 1) after each arc is added to the progression. Hence, we had expected negative cycles located by this algorithm to contain relatively few nodes. A possible explanation lies in the network generator. The direct search method will continue to add arcs to the progression as long as the sequence of partial sums remains negative. With the high percentage of negative arcs (50%) and the guaranteed path to node 1 (due to the arborescence), Florian's algorithm will continue to add arcs to the progression until this path or some other path to the home base is found.

The analysis of variance in Table 5-21 verifies that algorithm and nodes are the only main effects which significantly affect the number of nodes in the cycle. From the results of the previous two sections we conclude that an increase in the number of nodes in the network leads to an increase in the number of nodes in the negative cycle which leads to an increase in the number of arcs in the negative

Table 5-20. Mean Response--Number of Nodes in the Cycle

Algorithm		Nodes	
Yen	6.69	25	4.27
Bennington	4.68	50	7.10
Florian-Robert	9.53	75	7.50
Ford-Fulkerson	6.75	100	8.78

Table 5-21. Analysis of Variance--Number of Nodes in the Cycle

Source	Degrees of Freedom	Sum of Squares	Mean Square	F-ratio
Main Effects				
Algorithm	3	1718.1	572.7	32.50*
Nodes	3	1564.6	521.5	29.60*
Density	3	32.8	10.9	0.62
Arc Distribution	2	32.4	16.2	0.92
Interactions				
Algorithm-Nodes	9	351.3	39.0	2.21*
Algorithm-Density	9	1334.5	148.2	8.42*
Algorithm-Arc Distribution	6	95.6	15.9	0.90
Nodes-Density	9	267.4	29.7	1.68
Nodes-Arc Distribution	6	265.8	44.3	2.51*
Density-Arc Distribution	6	105.1	17.5	0.99
Error	519	9143.2	17.6	

*Significant at $\alpha = 0.05$ level

cycle and an associated increase in the value of the negative cycle (more negative). The foregoing statement implies that it is advantageous to have more arcs in the cycle which disagrees with intuition and the only explanation would be an excessive number of negative arcs in the network. An improved experimental design would include an additional parameter corresponding to the percentage of negative arcs in the network.

Reduction in the Percentage of Negative Arcs

Recall that the analysis from Table 5-4 found no significant effect on computational time due to the variance of the arc distribution. This variable could be dropped from the analysis and replaced with a variable D_i, which represents an upper bound on the percentage of negative arcs in the network.

An additional replication of the experiment was performed with the new independent variable in the mathematical model. D_i, i = 1,2,3,4,5 was analyzed at five fixed levels corresponding to 50,40,30,20 and 10% negative arcs, respectively. The associated analysis of variance with computational time as the response variable is presented in Table 5-22. The analysis did indeed conclude that the percentage of negative arcs in the network significantly affects the time required to locate a negative cycle. As the percentage of negative arcs is decreased, it becomes more

Table 5-22. Analysis of Variance--Computational Time Includes % of Negative Arcs

Source	Degrees of Freedom	Sum of Squares	Mean Square	F-ratio
Main Effects				
Algorithm	3	215.5	71.83	43.30*
Nodes	3	164.1	54.70	33.00*
Density	3	4.3	1.43	0.86
Arc Distribution	4	57.2	14.30	8.60*
Interactions				
Algorithm-Nodes	9	194.2	21.57	13.00*
Algorithm-Density	9	9.4	1.04	0.62
Algorithm-Arc Distribution	12	169.5	14.10	8.50*
Nodes-Density	9	20.5	2.30	1.40
Nodes-Arc Distribution	12	53.1	4.40	2.60*
Density-Arc Distribution	12	6.9	0.60	0.40
Error	243	402.8	1.66	

*Significant at $\alpha = 0.05$ level

Table 5-23. Analysis of Variance--Computational Time-Includes % of Negative Arcs-Excluding Florian and Robert's Algorithm

Source	Degrees of Freedom	Sum of Squares	Mean Square	F-ratio
Main Effects				
Algorithm	2	140.3	70.15	32.40*
Nodes	3	218.8	72.90	33.70*
Density	3	5.7	1.90	0.87
Arc Distribution	4	76.2	19.05	8.80*
Interactions				
Algorithm-Nodes	6	139.5	23.25	10.70*
Algorithm-Density	6	8.1	1.35	0.62
Algorithm-Arc Distribution	8	150.5	18.80	8.70*
Nodes-Density	9	27.3	3.00	1.40
Nodes-Arc Distribution	12	70.7	5.90	2.70*
Density-Arc Distribution	12	9.2	0.76	0.35
Error	174	375.9	2.16	

*Significant at $\alpha = 0.05$ level

difficult to locate a negative cycle.

The present analysis with the percentage of negative arcs included as an independent variable was rerun over three algorithms, excluding Florian and Robert's direct search method. The resulting ANOVA in Table 5-23 is almost identical with that in Table 5-22, where all four algorithms were analyzed.

A possible explanation for the significant algorithm effect is that the percentage of negative arcs affects some algorithms more drastically than others. An example is the Bennington algorithm where the number of pivots required to reach an optimal solution increases, possibly exponentially, as the percentage of negative arcs in the network decreases. Also, Florian and Robert's direct search method did not approach its theoretical upper bound on the number of required computations for any of the 80 networks examined. This contrasts the previous experiment in which two random networks caused considerable problems for the direct search method.

Regression Analysis

An attempt was made to fit a polynomial relationship to the data on computational time for each of the four algorithms. The model would be used for predictive purposes and to provide greater insight into the process of locating negative cycles. A 3rd degree polynomial of the form

$$Y = \beta_0 + \beta_1 N + \beta_2 N^2 + \beta_3 N^3 + \beta_4 D + \beta_5 D^2 + \beta_6 D^3 + \beta_7 DN$$

$$+ \beta_8 V + \beta_9 V^2 + \beta_{10} NV + \beta_{11} DV + \beta_{12} NDV$$

was chosen, where

N = # of nodes

D = density of the network

V = variance of the distribution from which the associated arc costs are generated.

The least squares estimates along with the associated square of the multiple correlation coefficient are presented in Table 5-24.

R^2 is defined as the proportion of the variation about the mean, explained by the model. An R^2 near 1 implies that almost all of the variation about the mean is accounted for by the model. Examination of the results in Table 5-24 shows a relatively poor fit for all algorithms with the exception of the Ford-Fulkerson algorithm which has an associated $R^2 = 0.81$.

An explanation is that the Ford-Fulkerson is a rather trivial algorithm consisting of three nested loops. An iteration of the Ford-Fulkerson algorithm consists in taking one node at a time and using its label in an attempt to improve the label on one of the remaining N-1 nodes. N iterations are usually required to determine the existence of a negative cycle. Therefore, the number of computations

Table 5-24. Least-Squares Estimators--Computational Time

Independent Variable		N	N^2	N^3	D	D^2	D^3
Algorithm	β_0	$\beta_1(1)$	$\beta_2(1)$	$\beta_3(2)$	β_4	β_5	β_6
Yen	0.36	95.00	-1.50	0.84	-10.42	70.27	-153.96
Bennington	-5.38	571.00	-7.70	6.20	127.17	-1046.74	2665.96
Florian	-12.81	13400.00	-220.00	110.00	-273.63	2082.49	-4627.52
Ford-Fulkerson	1.81	24.00	-6.80	6.20	-24.10	63.77	-102.03

Independent Variable	DN	V	V^2	NV	DV	NDV	
Algorithm	$\beta_7(1)$	$\beta_8(2)$	$\beta_9(3)$	$\beta_{10}(2)$	$\beta_{11}(2)$	$\beta_{12}(2)$	R^2
Yen	2.90	510.00	-0.99	-0.10	-9.70	0.52	0.17
Bennington	-970.00	54.00	2.60	-1.10	-290.00	6.00	0.42
Florian	-510.00	690.00	96.00	-7.30	-5400.00	43.00	0.09
Ford-Fulkerson	2800.00	6.80	-13.00	0.24	100.00	-1.10	0.81

(1) Entries are in 10^{-4}

(2) Entries are in 10^{-6}

(3) Entries are in 10^{-10}

N = # of nodes

D = density of the network

V = variance of the distribution from which the associated arc costs are generated

and comparisons required is roughly proportional to N^3 and can be fit rather well to a 3rd degree polynomial.

The performance of the remaining three algorithms is not as easy to explain. As stated earlier, the performance of Florian and Robert's algorithm is a function of the number of negative partial sums which can be formed around the home base node, while Yen's algorithm is based on the number of homogenous blocks in the shortest paths. Both of these terms, number of negative partial sums and the number of homogenous blocks are random variables. The time required for the Bennington algorithm to locate a negative cycle is a function of the initial arborescence and the manner in which arcs are chosen to enter the basis. Empirical evidence suggests that the number of pivots required to reach an optimal solution is also a random variable.

Further Research

To complete the research on randomly generated networks, we attempted to gain a measure of the relative difficulty involved in locating negative cycles compared to locating shortest paths. Since three of the four algorithms used to detect negative cycles are modifications of shortest path algorithms, the question arose as to which was the most difficult problem.

To obtain a sample of networks which did not contain any negative cycles, the upper bound on the percentage of

negative arcs was decreased to 5%. The solution times to determine the shortest paths on these networks were compared to the times required to detect a negative cycle on the identical networks when the percentage of negative arcs was increased to 50%.

The results show that locating negative cycles requires less computational effort than does locating shortest paths. The answer lies in the terminating criterion of the two problems.

Recall that during the kth iteration, the Ford-Fulkerson algorithm determines the optimal path from the source node to all nodes whose shortest path consists of no more than k arcs. During the same iteration, the Yen algorithm determines the shortest path from the origin to all nodes in the kth block of increasing and decreasing sequences. At least one node must become permanently labeled during each iteration in either algorithm. The reasoning behind this statement when applied to the Ford-Fulkerson algorithm is as follows: In order for a shortest path from the origin to some node y to contain k+1 arcs, there must exist a node x whose shortest path contains k arcs. The arc (x,y) will complete the shortest path to node y. An identical proof can be given for the Yen algorith, i.e. every block must contain at least one node. Therefore, both algorithms terminate with the optimal shortest paths whenever the node labels from the successive iterations are identical for all

nodes. Convergence is guaranteed in no more than N-1 iterations since no shortest path may contain more than N-1 arcs or blocks.

Termination with a negative cycle may occur as early as the 2nd iteration. A negative cycle is detected whenever the label on the source node becomes negative or a shortest path is found which contains more than N-1 arcs. The latter cannot be detected until the Nth iteration. Also, at the end of each iteration, we can inspect the node labels to assure that an additional node has become permanently labeled. If all node labels have been altered between the kth and k+1st iterations then no optimal path contains k+1 arcs nor k+1 blocks. Therefore, a negative cycle exists in the network.

Termination of the simplex algorithm with the shortest paths or a negative cycle is a function of the initial basic feasible solution and the manner in which nonbasic arcs are chosen to enter the basis. There is no evidence to indicate that this algorithm detects negative cycles "faster" than it locates shortest paths. The empirical results had already shown the Bennington algorithm to be the most severely affected algorithm in terms of computational time in locating negative cycles as the percentage of negative arcs in the network decreased. The average solution time over the five levels of density is presented in Table 5-25.

The reader must remember that our empirical results are based on networks containing at least some negative arc

Table 5-25. Average Computational Time--Bennington's Algorithm

Percentage Negative Arcs	Average Computational Time (sec)
50	0.65
40	0.62
30	1.29
20	1.71
10	5.11

distances. The superiority of these shortest path algorithms does not extend to the problem where arc distances are restricted to be nonnegative.

To substantiate this claim an additional set of test problems was solved for the shortest paths on non-negative cost networks using Yen's and Bennington's algorithms. The average computational times were 1.10 and 25.25 sec respectively. These outcomes suggest that the Bennington algorithm may not be an efficient shortest path algorithm.

Presently, the most efficient algorithm for the shortest path between a specified pair of nodes when the arc distances are restricted to be non-negative is the Dijkstra algorithm. If the shortest paths from the origin to all other nodes are desired, the algorithm requires N(N-1)/2 additions and N(N-1) comparisons to solve the problem. While empirical results on the Dijkstra algorithm were not available, its theoretical computational upper bound is less than that of our three shortest path algorithms. Consequently, it appears that the domain of efficiency for these algorithms is restricted to networks possessing at least some negative arcs.

Application to the Minimal Cost Flow Problem

The present chapter has reported computational experience in locating negative cycles on randomly generated networks. Once a negative cycle was found, the network was discarded and a new network created with different

characteristics. This evaluation may not be a true indication of how well an algorithm will perform when applied to locate negative cycles in a practical problem. Therefore, as a final evaluation, the four algorithms were employed to locate negative cycles in Klein's algorithm for minimal cost flow problems.

Over the set of problems solved, Bennington's algorithm produced the fastest average solution time. This can be attributed to the "restart" procedure which distinguishes the Bennington algorithm from the remaining algorithms which store no information from the previous cycles and must be "restarted" from scratch. By utilizing the "restart" technique, the initial arborescence in the Bennington algorithm need be created only once. After a negative cycle is located, an arborescence for the new marginal cost network is obtained from a modified version of the previous arborescence along with the reverse arcs from the negative cycle.

The computational experience showed that both the Yen and Ford-Fulkerson algorithms were required, on the average, to locate fewer negative cycles in a minimal cost flow problem than were the Bennington or Florian-Robert algorithms. An analysis also showed that the negative cycles detected by the Yen and Ford-Fulkerson algorithms were, on the average, "more negative" than the cycles located by the other two algorithms. Thus we have additional support for the hypothesis that an algorithm which concerns itself with locating cycles of large

negative value will require the tracing of fewer negative cycles in an iteration of Klein's algorithm than will an algorithm which does not concern itself with the value of a negative cycle. Although this is not a statistical test of hypotheses, the result is encouraging and warrants further research.

The computational experience with the Florian and Robert algorithm was disappointing. A number of networks arose which approached the theoretical upper bound of this algorithm.

Any difficulties encountered with this algorithm on random networks are accentuated in the minimal cost flow problem since the entire problem consists of examining similar networks. That is, if while solving a minimal cost flow problem using Klein's algorithm a network arises which causes Florian and Robert's algorithm to approach its computational upper bound, then the following network is also likely to present problems for the algorithm. The rationale is that once a negative cycle is identified on a marginal cost network, flow is sent around the cycle and the new marginal cost network is formed. The new marginal cost network will differ from the previous one in at most k ($2 \leq k \leq N$) arcs, i.e. those arcs corresponding to the negative cycle. Therefore, Klein's algorithm will retain a so called "bad" network until the minimal cost flow is obtained. This contrasts our experimental "runs" in which a network was discarded after a single negative cycle was located.

CHAPTER VI

CONCLUSIONS

The primary purpose of this thesis was to identify an efficient algorithm for locating negative cycles. We have accumulated both theoretical and empirical evidence which indicate that four valid algorithms are available. The algorithm which one chooses to implement will be a function of his particular problem as well as his programming ability.

The empirical results show the Florian and Robert direct search method to be superior to the remaining algorithms in locating negative cycles on random networks. We agree that an enormous computational upper bound does exist on this algorithm. This theoretical upper bound limits the application of the algorithm, especially in Klein's minimal cost flow algorithm, which retains the "basic" network throughout the entire problem. That is, when a marginal cost network presents a difficulty to the direct search method, the following marginal cost network (differing from the previous network in k arcs corresponding to the negative cycle) will also cause difficulties to Florian and Robert's algorithm.

Contrasting Florian and Robert's algorithm was the Bennington algorithm. The true capability of the latter was not realized until the algorithm was applied to the minimal

cost flow problem. The Bennington algorithm performed well in the experiment but was still rated third behind the Florian-Robert and Yen algorithms in terms of computational time and the quality of negative cycles located. This assumes that we disregard the two problems that caused excessive difficulty for the direct search method.

The Bennington algorithm is the only algorithm which retains information from previous cycles. This information was of no value to us in the experiment in which the network was discarded once a negative cycle was located. An algorithm like the Bennington algorithm, which possesses a "restart" procedure, is intuitively appealing and its relative worth was presented in the minimal cost flow problem in which it produced the lowest average time per problem solved.

The Yen algorithm was probably the overall most efficient of the four algorithms tested. It returned a negative cycle of reasonably high quality in a fairly low average computational time over the entire experiment. The algorithm was not severely affected by any particular combination of the independent factors comprising a network. The algorithm performed favorably in the minimal cost flow problem as well as in locating the shortest paths when negative costs were allowed. Yen's algorithm has the lowest theoretical upper bound on the number of required computations and this fact coupled with the empirical evidence recommends the algorithm as the most dependent. The only drawback would

appear to be in programming the algorithm for efficient use on the computer.

The Ford-Fulkerson algorithm was used primarily as a "control" algorithm against which to compare the remaining algorithms. The Ford-Fulkerson algorithm has withstood the test of time and produced respectable results in both the experiment and the minimal cost flow problem. The algorithm, while not providing the fastest computational time, did produce, on the average, the "most negative" cycle. Unlike the Yen algorithm, its strongest attribute is that the Ford-Fulkerson algorithm is easily programmed for implementation on the computer.

Our research has shown that the theoretical upper bound may not be a true indication of an algorithm's performance when test problems are generated at random. The theoretical upper bound of the Yen algorithm is half that of the Ford-Fulkerson algorithm, yet the empirical results showed the Yen algorithm to be 15 times faster than the Ford-Fulkerson. The Florian and Robert algorithm possessed the largest theoretical upper bound but this bound was realized on less than 3% of the test problems. Over the remaining random networks this algorithm performed consistently faster than any other algorithm. The immense theoretical upper bound on the Bennington algorithm was never attained. As noted in Chapter I and verified throughout the thesis, the theoretical upper bound is a conservative evaluation and

should not be the primary instrument by which to compare algorithms.

Extensions

Further work is suggested in the area, especially in the range of value of the negative cycle. While it is possible to find the "shortest" path, finding the "most negative" cycle seems extremely difficult. Part of the problem is that when negative cycles exist in the graph, the shortest paths are undefined. We are able to hypothesize that algorithms which produce cycles of large negative value are able to clear a marginal cost network of all cycles faster than algorithms which do not concern themselves with the value of the cycles. A formal proof of this proposition is desired and then an algorithm to locate the "most negative" cycle. Preliminary work in this area focusing primarily on the most negative node label and most negative marginal node label was unsuccessful. Research is also suggested on the relative merit of locating the negative cycle containing the largest number of arcs versus the cycle with the most negative value, with application of course, to the minimal cost flow problem.

The only obstacle preventing the Florian and Robert direct search method from surfacing as the most efficient algorithm to locate negative cycles is its computational upper bound. Therefore, a combination of this algorithm, together with a technique to partition the feasible solution space into

several subsets, might be prosperous. A branch-and-bound scheme is recommended.

APPENDICES

APPENDIX A

GENERATING M DISTINCT RANDOM ARCS

Recall that the procedure presented in Chapter IV to create an arborescence, generates only N-1 arcs. To obtain a network with a specified density d, (d*N*(N-1))-(N-1) additional arcs must be created. These arcs must be created so that no two arcs connect the same pair of nodes.

Initially, all possible arc combinations (i,j) were created and stored in an array WORD(10000) utilizing the following FORTRAN procedure.

ALGORITHM:

```
      k=1
      DO 100 I=1,N
      DO 100 J=1,N
      WORD(k)=1000 * I + J
      k=k+1
100   CONTINUE
```

The nodes can later be retrieved by the following procedure:

```
I(k) = WORD(k)/1000
J(k) = WORD(k)-(1000 * I(k))
```

When the first arc (i_1,j_1) was chosen for the network,

$$(i_1,j_1) = \text{WORD}(k) \text{ for some } k,$$
$$\text{call it } k^*$$

then WORD(k^*) is interchanged with WORD(1).

In general then, when the Lth arc is chosen for the network,

$$(i_L,j_L) = \text{WORD}(k) \text{ for some } k,$$
$$\text{call it } k^*$$

then WORD(k^*) is interchanged with WORD(L).

Therefore, at the end of iteration M, M arcs have been created and the pair of nodes (i,j) which are connected by each arc are stored in the first M positions of array WORD.

At the end of Step 4, the number of arcs which have been generated equals k = N-1. A random number is generated between k and N**2. This number acts as a pointer, and serves to specify and next arc to be added to the network. The generated number accomplishes this by pointing to an element in array WORD which contains two nodes not already joined by an arc. If arc k ($k \geq N-1$) is being generated, this element is interchanged with WORD(k), k is incremented by 1 and a new random number is generated between k and N**2 and serves to point to the next pair of nodes to be connected by an arc. At the end of iteration NREQ, the first NREQ places of array WORD contain the NREQ required arcs.

APPENDIX B

EXAMPLE PROBLEMS

Suppose the network and associated cost matrix are as in Fig. B-1. We will outline the steps involved in locating a negative cycle for each of the algorithms described in Chapter II.

Ford-Fulkerson's Algorithm

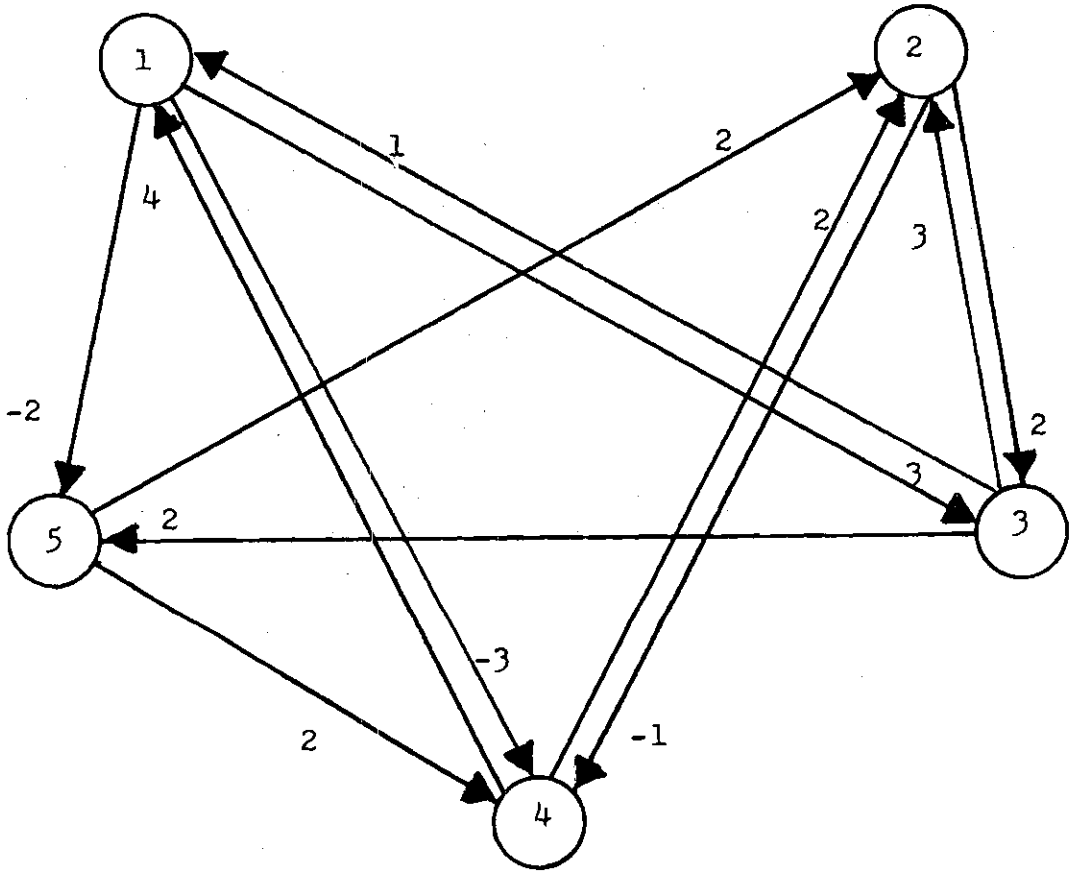
The labels on node j; j = 1,2,...,5, as they would be updated while examining node i; i = 1,2,...,5, are as follows:

$$\pi_1^{(0)} = [-,0], \quad \pi_2^{(0)} = \pi_3^{(0)} = \pi_4^{(0)} = \pi_5^{(0)} = [-,\infty]$$

Iteration 1

Node j

Node i	1	2	3	4	5
1	[-,0]	[-, ∞]	[1, 3]	[1,-3]	[1,-2]
2	[-,0]	[-, ∞]	[1, 3]	[1,-3]	[1,-2]
3	[-,0]	[3, 6]	[1, 3]	[1,-3]	[1,-2]
4	[-,0]	[4,-1]	[4,-2]	[1,-3]	[1,-2]
5	[-,0]	[4,-1]	[4,-2]	[1,-3]	[1,-2]



(a) A .60% Dense Network

	1	2	3	4	5
1	0	∞	3	-3	-2
2	∞	0	2	-1	∞
3	1	3	0	∞	2
4	4	2	∞	0	∞
5	∞	2	∞	2	0

(b) Associated Cost Matrix

Figure B-1. Example Network

Iteration 2

Node j

Node i	1	2	3	4	5
1	[-, 0]	[4,-1]	[4,-2]	[1,-3]	[1,-2]
2	[-, 0]	[4,-1]	[4,-2]	[1,-3]	[1,-2]
3	[3,-1]	[4,-1]	[4,-2]	[1,-3]	[1,-2]
4	[3,-1]	[4,-1]	[4,-2]	[1,-3]	[1,-2]
5	[3,-1]	[4,-1]	[4,-2]	[1,-3]	[1,-2]

Since the label on node 1 is negative, a negative cycle exists in the network. To obtain the nodes in the cycle, retrace the node labels.

Label(1) = 3

Label(3) = 4

Label(4) = 1

Label(1) = 3

Therefore the negative cycle is 3-1-4-3.

Yen's Modified Algorithm

The solution is obtained by applying the algorithm as follows:

Iteration 0--Obtain $\pi_i^{(0)}$, i = 1, 2, ..., 5 as follows:

$$\pi_1^{(0)} = c_{11} = 0, \text{ LABEL}(1) = 1$$

$$\pi_2^{(0)} = c_{12} = \infty, \text{ LABEL}(2) = 1$$

$$\pi_3^{(0)} = c_{13} = 3, \text{ LABEL}(3) = 1$$

$$\pi_4^{(0)} = c_{14} = -3, \text{LABEL}(4) = 1$$

$$\pi_5^{(0)} = c_{15} = -2, \text{LABEL}(5) = 1$$

Iteration 1--Starting from i=1 toward i=5 compute $\pi_i^{(1)}$, i = 1, 2, ..., 5 as follows:

$$\pi_1^{(1)} = \pi_1^{(0)} = 0$$

$$\text{LABEL}(1) = 1$$

$$\pi_2^{(1)} = \min[\pi_1^{(1)} + c_{12}, \pi_2^{(0)}] = \infty$$

$$\text{LABEL}(2) = 1$$

$$\pi_3^{(1)} = \min[\pi_1^{(1)} + c_{13}, \pi_2^{(1)} + c_{23}, \pi_3^{(0)}] = 3$$

$$\text{LABEL}(3) = 1$$

$$\pi_4^{(1)} = \min[\pi_1^{(1)} + c_{14}, \pi_2^{(1)} + c_{24}, \pi_3^{(1)} + c_{34}, \pi_4^{(0)}] = -3$$

LABEL(4) = 1

$$\pi_5^{(1)} = \min[\pi_1^{(1)} + c_{15},\ \pi_2^{(1)} + c_{25},\ \pi_3^{(1)} + c_{35},\ \pi_4^{(1)} + c_{45},\ \pi_5^{(0)}] = -2$$

LABEL(5) = 1

Iteration 2--Starting from i=5 toward i=1 compute $\pi_i^{(2)}$, i = 5, 4, ..., 1, as follows:

$$\pi_5^{(2)} = \pi_5^{(1)} = -2$$

LABEL(5) = 1

$$\pi_4^{(2)} = \min[\pi_5^{(2)} + c_{54}, \pi_4^{(2)}] = -3$$

LABEL(4) = 1

$$\pi_3^{(2)} = \min[\pi_5^{(2)} + c_{53},\ \pi_4^{(2)} + c_{43},\ \pi_3^{(1)}] = -2$$

LABEL(3) = 4

$$\pi_2^{(2)} = \min[\pi_5^{(2)} + c_{52},\ \pi_4^{(2)} + c_{42},\ \pi_3^{(2)} + c_{32},\ \pi_2^{(1)}] = -1$$

LABEL(2) = 4

$$\pi_1^{(2)} = \min[\pi_5^{(2)} + c_{51},\ \pi_4^{(2)} + c_{41},\ \pi_3^{(2)} + c_{31},\ \pi_2^{(1)} + c_{21},$$
$$\pi_1^{(1)}] = -1$$

LABEL(1) = 3

Since $\pi_1^{(2)} = -1 < 0$, the search is terminated with a negative cycle. To obtain the nodes in the cycle, retrace the labels.

LABEL(1) = 3

LABEL(3) = 4

LABEL(4) = 1

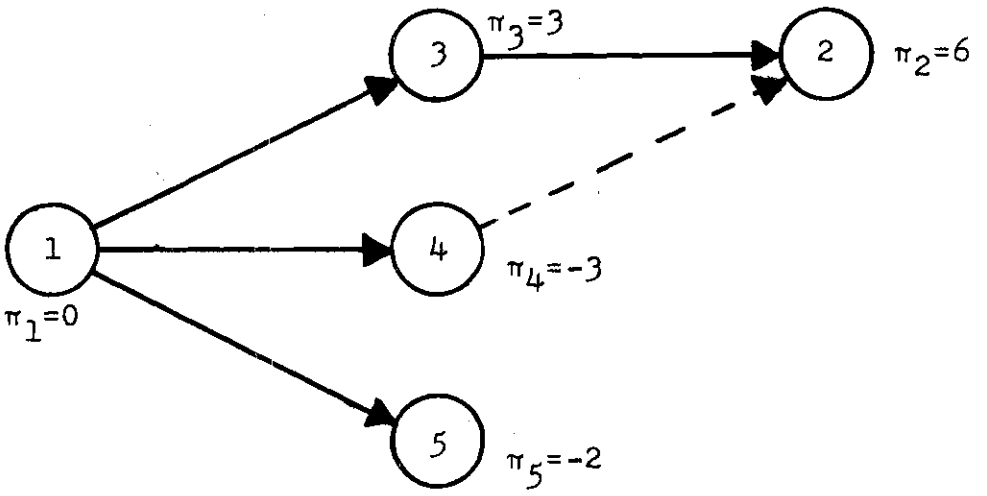
LABEL(1) = 3

Therefore the negative cycle is 3-1-4-3.

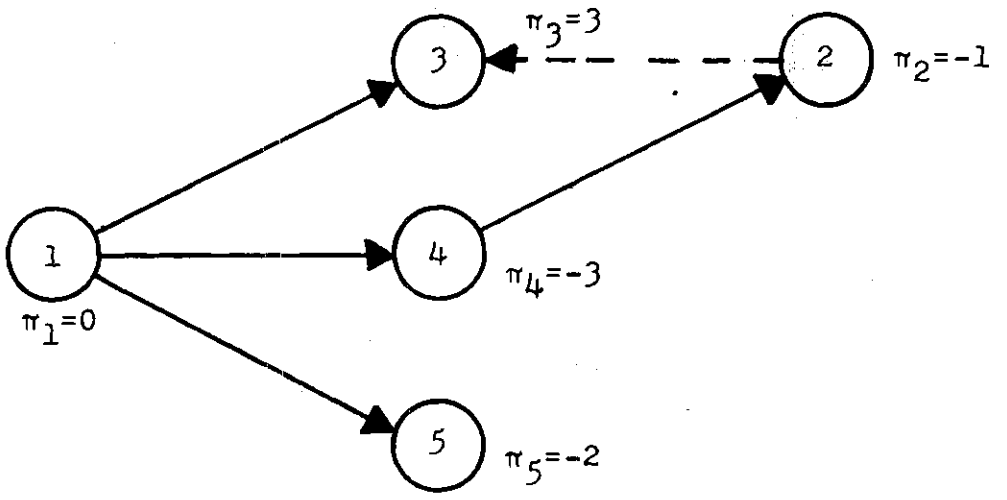
Bennington's Simplex Algorithm

Iteration 1

Suppose the initial arborescence is as in Fig. B-2(a). The corresponding simplex multipliers are shown alongside each node. Since $\pi_4 + c_{42} < \pi_2$, arc (4,2) will enter the basis. The non-basic arc entering the basis is shown as a dashed line in Fig. B-2(a). Since the path from node 1 to node 4 does not include node 2, we have not found a negative cycle. Arc (4,2) enters the basis and arc (3,2) leaves the basis.



(a) Iteration 1



(b) Iteration 2

Figure B-2. Basic Feasible Solutions

Iteration 2

The current solution along with the simplex multipliers are shown in Fig. B-2(b). Since $\pi_2 + c_{23} < \pi_3$, the non-basic (dashed) arc (2,3) is chosen to enter the basis. The path from node 1 to node 2 does not contain node 3. Therefore, we have not found a negative cycle. Arc (2,3) enters the basis and arc (1,3) leaves the basis.

Iteration 3

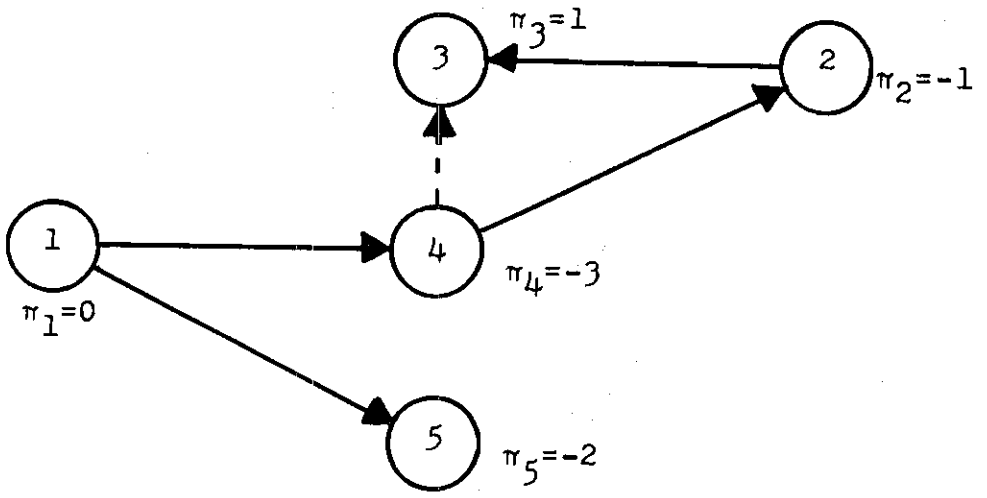
The current solution along with simplex multipliers is shown in Fig. B-3(a). Since $\pi_4 + c_{43} < \pi_3$, arc (4,3) is selected to enter the basis. The path from node 1 to node 4 does not include node 3. Therefore, we have not found a negative cycle. Arc (4,3) enters the basis and arc (1,3) leaves the basis.

Iteration 4

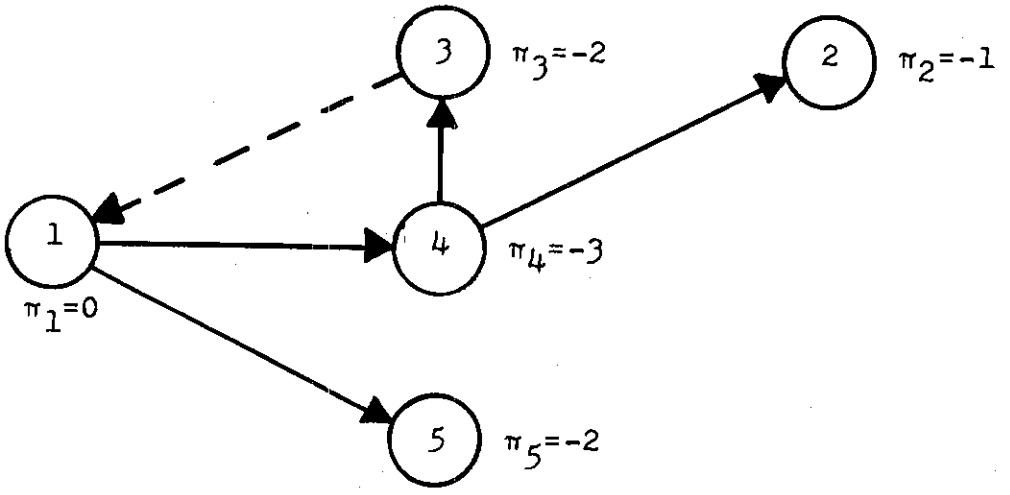
Since $\pi_3 + c_{31} < \pi_1$, arc (3,1) will enter the basis. This arc is shown as the dashed arc in Fig. B-3(b), which is the current solution. The path from node 1 to node 3 includes node 3. Therefore, the arc (3,1) along with the path from node 1 to node 3, i.e. 1-4-3, will form a negative cycle. The negative cycle is 1-4-3-1.

Florian and Robert Direct Search Method

0. Search for a sequence of negative partial sums starting from node 1.
1. Search along row 1 from column 1 to column 5 to locate the first negative element.



(a) Iteration 3



(b) Iteration 4

Figure B-3. Basic Feasible Solutions

2. Search is successful. Node 4 is added to the sequence. The sequence thus far is 1-4. The value of this sequence is $c_{14} = -3$.
3. Test whether the cycle can be completed. Since $-3 + c_{41} \not< 0$, the cycle cannot be completed.
4. Search along row 4 from column 1 to column 5 to locate an element v not already in the sequence, such that $-3 + c_{4v} < 0$.
5. Search is successful. Node 2 is added to the sequence. The sequence thus far is 1-4-2. The value of this sequence is $c_{14} + c_{42} = -1$.
6. Test whether the cycle can be completed. Since $-1 + c_{21} \not< 0$ the cycle cannot be completed.
7. Search along row 2 from column 1 to column 5 to locate an element v not already in the sequence such that $-1 + c_{2v} < 0$.
8. Search is unsuccessful. Therefore we must backtrack. Backtrack to node 4 and the sequence is 1-4. The value of the sequence is $c_{14} = -3$.
9. Search along row 4 from column 3 to column 5 to locate an element v not already in the sequence such that $-3 + c_{4v} < 0$.
10. Search is successful. Node 3 is added to the sequence. The sequence thus far is 1-4-3. The value of the sequence is $c_{14} + c_{43} = -2$.
11. Test whether the cycle can be completed. Since $-2 + c_{31} =$

$-1 < 0$, the cycle has been completed. The negative cycle is 1-4-3-1.

APPENDIX C

PROGRAM LISTINGS

A source program listing is contained in the appendix.

1. The Yen Algorithm
2. The Florian and Robert Algorithm
3. The Bennington Algorithm
4. The Ford-Fulkerson Algorithm

Program Notation

N = Number of nodes

L = Initial node in the negative cycle

IFEAS = 0, the network contains a negative cycle

= 1, the network does not contain a negative cycle

JSUM = The sum of arc costs around the cycle

The jth arc is directed from AYE(J) to JAY(J).

The cost associated with the jth arc is ARC(J).

The arcs are sorted with a major sort on the beginning node i and a minor sort, within each i, on the terminating node j. In this sorted list, the arcs leaving node i are G(i) through G(i+1)-1 in AYE.

JPRIME contains the list of arcs sorted with a major sort on the terminating node j and a minor sort, within each j, on the beginning node i. The arcs entering node j are H(j) through H(j+1)-1 in JPRIME.

In the Bennington algorithm, NREQ is the number of arcs in the network.

```
      SUBROUTINE SPATH(N,L,IFEAS,JSUM)
      DIMENSION IFUNC(100,3),ITEMP(100),IPOS(100,3)
      DIMENSION JPOS(100),JF(100),ISAVE(100)
      DIMENSION TCYC(100),ISTORE(100)
      INTEGER AYE,G,H,CYCLE,TCYC
      REAL JF,IFUNC,ITEMP,ISAVE,JSUM
      COMMON/BLOCK1/ AYE(2050),JAY(2050),JPRIME(2050)
      COMMON/BLOCK2/ WORD(10000)
      COMMON/BLOCK3/ G(101),H(101),CYCLE(100),ARC(2050)
C******YEN'S MODIFIED ALGORITHM
      WRITE(6,4)
4     FORMAT(1X,'YEN'S MODIFIED ALGORITHM')
C******CREATE JF(I)=D(1,I)
      JF(1)=0.
      DO 1 I=2,N
1     JF(I)=9999.
      NL=G(2)-1
      IF(NL.LT.1)GO TO 3


      DO 2 I=1,NL
2     JF(JAY(I))=ARC(I)
C*****PERFORM INITIAL ITERATIONS
3     NUM=N-1
C*****ITERATION #1
      IFUNC(1,1)=JF(1)
      IPOS(1,1)=1
      DO 40 I=2,N
      JJ=0
      IF(H(I).EQ.0.OR.H(I+1)-1.LT.H(I))GO TO 31
C*****ARCS INTO I ARE H(I) THROUGH H(I+1)-1 IN JPRIME
      NK=H(I)
      NL=H(I+1)-1
      DO 30 J=NK,NL
C*****ONLY LOOK AT THOSE D(J,I) WITH J .LT. I
      IF(AYE(JPRIME(J)).GE.I)GO TO 31
      JJ=JJ+1
      ITEMP(JJ)=IFUNC(AYE(JPRIME(J)),1)+ARC(JPRIME(J))
30    JPOS(JJ)=AYE(JPRIME(J))
      CALL MIN(ITEMP,JPOS,1,JJ,M)
      IFUNC(I,1)=ITEMP(1)
      IPOS(I,1)=JPOS(1)
      IF(IFUNC(I,1).LT.JF(I))GO TO 40
31    IFUNC(I,1)=JF(I)
      IPOS(I,1)=1
40    CONTINUE
C     WRITE(6,15)(IFUNC(K,1),K=1,N)
15    FORMAT(1X/,1X,5F10.2)
C     WRITE(6,16)(IPOS(K,1),K=1,N)
16    FORMAT(1X,5I3)
C*****ITERATION # 2
      IFUNC(N,2)=IFUNC(N,1)
```

```
      IPOS(N,2)=IPOS(N,1)
      DO 400 I=1,NUM
      M=N-I
      JJ=0
      IF(H(M).EQ.0.OR.H(M+1)-1.LT.H(M))GO TO 302
      NK=H(M)
      NL=H(M+1)-1
      NL=NL+1
      DO 300 L=1,2000
      J=NL-L
      IF(J.LT.NK.OR.AYE(JPRIME(J)).LE.M)GO TO 301
      JJ=JJ+1
      ITEMP(JJ)=IFUNC(AYE(JPRIME(J)),2)+ARC(JPRIME(J))
300   JPOS(JJ)=AYE(JPRIME(J))
301   CALL MIN(ITEMP,JPOS,1,JJ,N)
      IFUNC(M,2)=ITEMP(1)
      IPOS(M,2)=JPOS(1)
      IF(IFUNC(M,2).LT.IFUNC(M,1))GO TO 400
302   IFUNC(M,2)=IFUNC(M,1)
      IPOS(M,2)=IPOS(M,1)
400   CONTINUE
      KKK=2
C*****CHECK CONVERGENCE
C     WRITE(6,15)(IFUNC(K,2),K=1,N)
C     WRITE(6,16)(IPOS(K,2),K=1,N)
      K=2
      IF(IFUNC(1,2).LT.0.)GO TO 760
C*****PERFORM 3RD ITERATION
      K=3
      L=0


      IFUNC(1,3)=IFUNC(1,2)
      IPOS(1,3)=IPOS(1,2)
      IF(IFUNC(1,3).EQ.IFUNC(1,1))GO TO 71
      L=1
      ISTORE(L)=1
71    DO 410 I=2,N
      IF(L.EQ.0)GO TO 310
74    JJ=0
      DO 331 J=1,L
      IF(G(ISTORE(J)).EQ.0.OR.G(ISTORE(J)+1)-1
     6.LT.G(ISTORE(J)))GO TO 331
      NK=G(ISTORE(J))
      NL=G(ISTORE(J)+1)-1
      DO 332 JK=NK,NL
      IF(JAY(JK).EQ.I)GO TO 76
332   IF(JAY(JK).GT.I)GO TO 331
      GO TO 331
78    JJ=JJ+1
      ITEMP(JJ)=IFUNC(ISTORE(J),3)+ARC(JK)
      JPOS(JJ)=ISTORE(J)
```

```
331     CONTINUE
        CALL MIN(ITEMP,JPOS,1,JJ,N)
        IFUNC(I,3)=ITEMP(1)
        IPOS(I,3)=JPOS(1)
        IF(IFUNC(I,3).LT.IFUNC(I,2))GO TO 309
310     IPOS(I,3)=IPOS(I,2)
        IFUNC(I,3)=IFUNC(I,2)
309     IF(IFUNC(I,3).EQ.IFUNC(I,1))GO TO 410
        L=L+1
        ISTORE(L)=I
410     CONTINUE
        KKK=3
C       WRITE(6,15)(IFUNC(J,3),J=1,N)
C       WRITE(6,16)(IPOS(J,3),J=1,N)
        GO TO 600
C*****PERFORM REMAINING ITERATIONS
C*****L=# OF UPDATED VARIABLES
405     L=0
        DO 502 I=1,N
        IFUNC(I,1)=IFUNC(I,2)
        IPOS(I,1)=IPOS(I,2)
        IFUNC(I,2)=IFUNC(I,3)
502     IPOS(I,2)=IPOS(I,3)
        IF((K/2)*2.EQ.K)GO TO 100
C*****ODD ITERATION
        IF(IFUNC(1,3).EQ.IFUNC(1,1))GO TO 70
        L=1
        ISTORE(L)=1
70      DO 4000 I=2,N
        IF(L.EQ.0)GO TO 3040
75      JJ=0
        DO 341 J=1,L
        IF(G(ISTORE(J)).EQ.0.OR.G(ISTORE(J)+1)-1
       6.LT.G(ISTORE(J)))GO TO 341
        NK=G(ISTORE(J))
        NL=G(ISTORE(J)+1)-1
        DO 3332 JK=NK,NL
        IF(JAY(JK).EQ.I)GO TO 780
3332    IF(JAY(JK).GT.I)GO TO 341
        GO TO 341
780     JJ=JJ+1
        ITEMP(JJ)=IFUNC(ISTORE(J),3)+ARC(JK)


        JPOS(JJ)=ISTORE(J)
341     CONTINUE
        CALL MIN(ITEMP,JPOS,1,JJ,N)
        IFUNC(I,3)=ITEMP(1)
        IPOS(I,3)=JPOS(1)
        IF(IFUNC(I,3).LT.IFUNC(I,2))GO TO 3090
        IPOS(I,3)=IPOS(I,2)
        IFUNC(I,3)=IFUNC(I,2)
```

```
3090   IF(IFUNC(I,3).EQ.IFUNC(I,1))GO TO 4000
       L=L+1
       ISTORE(L)=I
4000   CONTINUE
C      WRITE(6,15)(IFUNC(J,3),J=1,N)
C      WRITE(6,16)(IPOS(J,3),J=1,N)
       GO TO 600
C*****EVEN ITERATION
100    IF(IFUNC(N,3).EQ.IFUNC(N,1))GO TO 7800
       L=1
       ISTORE(L)=N
7800   DO 800 I=1,NUM
       M=N-I
       JJ=0
       IF(L.EQ.0)GO TO 4095
       DO 3041 J=1,L
       IF(G(ISTORE(J)).EQ.0.OR.G(ISTORE(J)+1)-1
     6.LT.G(ISTORE(J)))GO TO 3041
       NK=G(ISTORE(J))
       NL=G(ISTORE(J)+1)
       DO 3322 JK=1,2000
       JL=NL-JK
       IF(JL.LT.NK.OR.JAY(JL).LT.M)GO TO 3041
3322   IF(JAY(JL).EQ.M)GO TO 785
       GO TO 3041
785    JJ=JJ+1
       ITEMP(JJ)=IFUNC(ISTORE(J),3)+ARC(JL)
       JPOS(JJ)=ISTORE(J)
3041   CONTINUE
       CALL MIN(ITEMP,JPOS,1,JJ,N)
       IFUNC(M,3)=ITEMP(1)
       IPOS(M,3)=JPOS(1)
       IF(IFUNC(M,3).LT.IFUNC(M,2))GO TO 4095
4090   IFUNC(M,3)=IFUNC(M,2)
       IPOS(M,3)=IPOS(M,2)
4095   IF(IFUNC(M,3).EQ.IFUNC(M,1))GO TO 800
       L=L+1
       ISTORE(L)=M
800    CONTINUE
C      WRITE(6,15)(IFUNC(J,3),J=1,N)
C      WRITE(6,16)(IPOS(J,3),J=1,N)
C*****CHECK IF WE HAVE CONVERGED
C*****FIRST TERMINATING CRITERION
600    IF(IFUNC(1,3).LT.0)GO TO 760
C*****2ND TERMINATING CRITERION
C*****N-L=#OF IFUNC(I,K)=IFUNC(I,K)=IFUNC(I,K-2)
       NU=N-L
       KK=K-2
       IF(NU.LT.KK)GO TO 760
C*****3RD TERMINATING CRITERION
```

```
      DO 650 KOUNT=1,N
650   IF(IFUNC(KOUNT,3).NE.IFUNC(KOUNT,1))GO TO 500

C*****CONVERGED-NO NEGATIVE CYCLES
      IFEAS=1
      RETURN
500   K=K+1
      GO TO 405
C*****A NEGATIVE CYCLE EXISTS
760   KOUNT=N
      IK=N
761   TCYC(N)=IPOS(KOUNT,KKK)
      IF(TCYC(N).EQ.1.AND.IPOS(1,KKK).EQ.1)GO TO 700
      NN=N-1
      DO 200 I=1,NN
      M=N-I
      KOUNT=IPOS(TCYC(M+1),KKK)
      IF(KOUNT.EQ.TCYC(M))GO TO 201
      TCYC(M)=KOUNT
      IF(TCYC(M).EQ.1.AND.IPOS(1,KKK).EQ.1)GO TO 700
200   CONTINUE
      JSUM=0.
      CYCLE(1)=TCYC(1)
      L=1
      DO 317   J=2,N
      IF(TCYC(J).EQ.CYCLE(1))GO TO 318
      L=L+1
      CYCLE(J)=TCYC(J)
317   CONTINUE
318   CONTINUE
      GO TO 319
201   JSUM=0.
      L=1
      M=M+1
      CYCLE(1)=TCYC(M)
      M=M+1
      DO 210 J=M,N
      L=L+1
      CYCLE(L)=TCYC(J)
C     JSUM=JSUM+COST(CYCLE(L-1),CYCLE(L))
210   CONTINUE
C     JSUM=JSUM+COST(CYCLE(L),CYCLE(1))
319   CONTINUE
511   FORMAT(2X,*JSUM= *,F13.2)
      WRITE(6,*)(CYCLE(J),J=1,L)
      WRITE(6,513)K
513   FORMAT(1X,*NUMBER OF ITERATIONS= *,I3)
      IFIN=CYCLE(1)
      IFEAS=0
      RETURN
700   IF(IK.EQ.1)GO TO 751
```

```
C
C
      IK=IK-1
      KOUNT=IK
      GO TO 761
751   IFEAS=1
      RETURN
      END
```

```
      SUBROUTINE MIN(ITEMP,JPOS,J,K,MZM)
      DIMENSION ITEMP(100),JPOS(100)
      REAL ITEMP
      COMMON/BLOCK1/ AYE(2050),JAY(2050),JPRIME(2050)
      COMMON/BLOCK2/ WORD(10000)
      COMMON/BLOCK3/ G(101),H(101),CYCLE(100),ARC(2050)


      IF(K.EQ.0)GO TO 3
      IF(K.EQ.1)RETURN
      I=J+1
      DO 2 M=I,K
      IF(ITEMP(M).GE.ITEMP(J))GO TO 2
      ITEMP(J)=ITEMP(M)
      JPOS(J)=JPOS(M)
2     CONTINUE
      RETURN
3     ITEMP(J)=999999.
      RETURN
      END
```

```
C
      SUBROUTINE FLORIAN(N,L,IFEAS,JSUM)
      DIMENSION VALUE(100),OPEN(100)
      INTEGER CYCLE,G,OPEN,L,S,U,W
      REAL JSUM
      COMMON/BLOCK1/ AYE(2050),JAY(2050),JPRIME(2050)
      COMMON/BLOCK2/ WORD(10000)
      COMMON/BLOCK3/ G(101),H(101),CYCLE(100),ARC(2050)
C*****FLORIAN AND ROBERT'S ALGORITHM
      WRITE(6,4)
4     FORMAT(1X,*FLORIAN AND ROBERT'S  ALGORITHM*)
      I=1
10    DO 20 J=1,N
      CYCLE(J)=0
      VALUE(J)=0.
20    OPEN(J)=1
      OPEN(I)=0
      VALUE(1)=0.
      CYCLE(1)=I
      L=1
      U=I
      IF(G(U).EQ.0.OR.G(U+1)-1.LT.G(U))GO TO 70
      S=G(U)
      NN=G(U+1)-1
29    DO 30 J=S,NN
      VAL=(OPEN(JAY(J))*(VALUE(L)+ARC(J)))
      IF(VAL.LT.0.)GO TO 50
30    CONTINUE
      GO TO 70
50    OPEN(JAY(J))=0
      L=L+1
      CYCLE(L)=JAY(J)
601   FORMAT(1X,*CYCLE(*,I2,*)= *,I3)
      VALUE(L)=VAL
      U=JAY(J)
      W=CYCLE(1)
C*****LOOK AT ALL ARCS OUT OF U
C*****TO SEE IF THERE IS AN ARC (U,W)
C
C*****IF NO ARC LEAVING U GO TO 70
      IF(G(U).EQ.0.OR.G(U+1)-1.LT.G(U))GO TO 70
      S=G(U)
      NN=G(U+1)-1
      DO 68 K=S,NN
      IF(JAY(K).EQ.W)GO TO 69
68    IF(JAY(K).GT.W)GO TO 29
      GO TO 29
69    IF((VALUE(L)+ARC(K)).LT.0.)GO TO 100
          GO TO 29
70    IF(L.EQ.1)GO TO 80
      S=CYCLE(L)
```

```
C
      VALUE(L)=0.
      OPEN(S)=1
      IF(S.EQ.JAY(G(CYCLE(L-1)+1)-1))GO TO 75
      NK=G(CYCLE(L-1))
      NL=G(CYCLE(L-1)+1)-1
      DO 71 JJ=NK,NL
71    IF(S.EQ.JAY(JJ))GO TO 72
      STOP
72    S=JJ+1
      NN=NL
      L=L-1
      U=CYCLE(L)
      GO TO 29
75    L=L-1
      GO TO 70
80    IF(I.EQ.N)GO TO 120
      I=I+1
      GO TO 10
100   IFIN=CYCLE(L)
      IFEAS=0
      WRITE(6,*)(CYCLE(J),J=1,L)
      RETURN
120   IFEAS=1
      RETURN
      END
```

```
      SUBROUTINE BENN(N,L,IFEAS,JSUM,NREQ)
      DIMENSION NODE(100),ILABL(100),LABEL(100)
      DIMENSION U(100),LNODE(100),TCYCLE(100)
      INTEGER TCYCLE,CYCLE,G,H,AYE
      REAL JSUM
      COMMON/BLOCK1/ AYE(2050),JAY(2050),JPRIME(2050)
      COMMON/BLOCK2/ WORD(10000)
      COMMON/BLOCK3/ G(101),H(101),CYCLE(100),ARC(2050)
C*****BENNINGTON'S ALGORITHM
      WRITE(6,4)
4     FORMAT(1X,*BENNINGTON'S ALGORITHM*)
      NPIV=0
C*****DETERMINE ARBORESCENCE CENTERED AT NODE1
C*****USING ARTIFICIAL ARCS IF NECESSARY
      U(1)=0.
      DO 5 I=1,N
5     ILABL(I)=0
      DO 6 I=1,N
      IF(G(I).GT.0)GO TO 7
      LABEL(I+1)=I
      U(I+1)=U(I)+9999.
6     ILABL(I)=1
C*****DEFINE ILABL(I)
C*****ILABL(I)=0 MEANS NODE I UNLABELED
C*****ILABL(I)=-1 MEANS NODE I LABELED BUT UNSCANNED
C*****ILABL(I)=1 MEANS NODE I LABELED AND SCANNED
7     NK=G(I)
      NL=G(I+1)-1
      DO 8 L=NK,NL
      IF(ILABL(JAY(L)).NE.0)GO TO 8
      ILABL(JAY(L))=-1
      LABEL(JAY(L))=I
      U(JAY(L))=U(I)+ARC(L)
8     CONTINUE
      ILABL(I)=1


C*****FIND A LABELED UNSCANNED NODE
12    DO 9 I=1,N
9     IF(ILABL(I).EQ.-1)GO TO 11
C*****ALL NODES HAVE BEEN SCANNED
      DO 10 I=1,N
      IF(ILABL(I).NE.0)GO TO 10
      LABEL(I)=1
      U(I)=9999.
10    CONTINUE
      GO TO 59
11    ILABL(I)=1
      IF(G(I).EQ.0.OR.G(I+1)-1.LT.G(I))GO TO 12
      NK=G(I)
      NL=G(I+1)-1
      DO 13 L=NK,NL
```

```
      IF(ILABL(JAY(L)).NE.0)GO TO 13
      ILABL(JAY(L))=-1
      LABEL(JAY(L))=I
      U(JAY(L))=U(I)+ARC(L)
13    CONTINUE
      GO TO 12
C*****COMPUTE SIMPLEX MULTIPLIERS
30    DO 31 K=1,N
31    NODE(K)=0
      U(1)=0.
      INDEX=0
      K=1
      LNODE(K)=1
42    INDEX=INDEX+1
      I=LNODE(INDEX)
      DO 40 J=2,N
      IF(LABEL(J).NE.I)GO TO 45
C*****NODE J WAS LABELED FROM NODE I
C*****FIND ARC(I,J)
C*****NOTE: THERE MIGHT NOT BE AN ARC SINCE
C*****(I,J) COULD BE IN ARTIFICIAL ARBORESCENCE
      IF(G(I).EQ.0.OR.G(I+1)-1.LT.G(I))GO TO 45
      NK=G(I)
      NL=G(I+1)-1
      DO 43 L=NK,NL
      IF(JAY(L).GT.J)GO TO 45
43    IF(JAY(L).EQ.J)GO TO 44
      GO TO 45
44    U(J)=U(I)+ARC(L)
      GO TO 46
45    U(J)=U(I)+9999.
46    NODE(J)=J
602   FORMAT(1X,*U(*,I2,*)= *,F10.2)
      K=K+1
      LNODE(K)=J
40    CONTINUE
C*****CHECK IF ALL MULTIPLIERS HAVE BEEN COMPUTED
      DO 41 J=2,N
41    IF(NODE(J).EQ.0)GO TO 42
C*****SELECT AN ARC TO ENTER BASIS
59    DO 60 I=1,NREQ
      IF(U(AYE(I))+ARC(I).GE.U(JAY(I)))GO TO 60
C*****ARC(I) WILL ENTER BASIS
      NPIV=NPIV+1
C     WRITE(6,600)AYE(I),JAY(I)
600   FORMAT(1X,*ARC(*,I2,*,*,I2,*)WILL ENTER*)
      GO TO 70


60    CONTINUE
C*****WE ARE OPTIMAL(NO NEGATIVE CYCLES)
      IFEAS=1
```

```
      RETURN
70    NEXT=AYE(I)
      IF(NEXT.EQ.1)GO TO 90
      TCYCLE(N)=AYE(I)
      K=0
80    NEXT=LABEL(NEXT)
      K=K+1
      TCYCLE(N-K)=NEXT
      IF(NEXT.EQ.JAY(I))GO TO 100
      IF(NEXT.EQ.1)GO TO 90
      GO TO 80
90    LABEL(JAY(I))=AYE(I)
C     WRITE(6,601)JAY(I),LABEL(JAY(I))
601   FORMAT(1X,*LABEL(*,I2,*)= *,I2)
      GO TO 30
100   CYCLE(1)=TCYCLE(N-K)
      L=1
      IFEAS=0
      DO 110 I=2,N
      CYCLE(I)=TCYCLE(N-K+I-1)
      L=L+1
      IF(N-K+I-1.EQ.N)GO TO 120
110   CONTINUE
120   WRITE(6,*)(CYCLE(I),I=1,L)
      WRITE(6,604)NPIV
604   FORMAT(1X,*NO. PIVOTS = *,I6)
      RETURN
      END
```

```
      SUBROUTINE FORD(N,L,IFEAS,JSUM)
      DIMENSION TCYCLE(100),STORE(100)
      DIMENSION PI(100,2),LABEL(100)
      INTEGER TCYCLE,CYCLE,G,H,AYE
      REAL JSUM
      COMMON/BLOCK1/ AYE(2050),JAY(2050),JPRIME(2050)
      COMMON/BLOCK2/ WORD(10000)
      COMMON/BLOCK3/ G(101),H(101),CYCLE(100),ARC(2050)
C*****FORD-FULKERSON'S ALGORITHM
C*****ASSIGN INITIAL LABELS
      WRITE(6,4)
4     FORMAT(1X,*FORD-FULKERSON'S ALGORITHM*)
      PI(1,1)=PI(1,2)=0.
      DO 1 I=2,N
1     PI(I,1)=PI(I,2)=9999.
C*****PERFORM ITERATION #1
      DO 6 I=1,N
C*****CHECK FOR ARCS OUT OF I
      IF(G(I).EQ.0.OR.G(I+1)-1.LT.G(I))GO TO 6
      DO 5 J=1,N
C*****CHECK FOR ARC(I,J)
      NK=G(I)
      NL=G(I+1)-1
      DO 7 K=NK,NL
      IF(JAY(K).GT.J)GO TO 5
7     IF(JAY(K).EQ.J)GO TO 8
      GO TO 5
8     IF(PI(I,2)+ARC(K).GE.PI(J,2))GO TO 5
      PI(J,2)=PI(I,2)+ARC(K)


      LABEL(J)=I
5     CONTINUE
6     CONTINUE
C*****PERFORM REMAINING N-1 ITERATIONS
      ITEST=N-1
      DO 200 ITER=2,N
      DO 51 I=1,N
C*****TRY TO IMPROVE LABEL ON NODE I
      DO 50 J=1,N
C*****CHECK FOR ARCS OUT OF NODE J
      IF(G(J).EQ.0.OR.G(J+1)-1.LT.G(J))GO TO 50
C*****CHECK FOR ARC(J,I)
      NK=G(J)
      NL=G(J+1)-1
      DO 70 K=NK,NL
      IF(JAY(K).GT.I)GO TO 50
70    IF(JAY(K).EQ.I)GO TO 80
      GO TO 50
80    IF(PI(J,2)+ARC(K).GE.PI(I,2))GO TO 50
      PI(I,1)=PI(I,2)
      PI(I,2)=PI(J,2)+ARC(K)
```

```
C
C
      LABEL(I)=J
50    CONTINUE
51    CONTINUE
      I=1
      IF(PI(1,2).LT.0.)GO TO 300
      IF(ITER.NE.ITEST)GO TO 200
      DO 60 I=1,N
60    STORE(I)=PI(I,2)
200   CONTINUE
C*****TEST FOR EXISTENCE OF NEGATIVE CYCLE
      DO 250 I=1,N
250   IF(PI(I,2).NE.STORE(I))GO TO 300
C*****NO NEGATIVE CYCLES
      IFEAS=1
      RETURN
300   NEXT=I
      TCYCLE(1)=I
C     WRITE(6,610)TCYCLE(1)
610   FORMAT(1X,I4)
      L=1
      IFEAS=0
325   NEXT=LABEL(NEXT)
      DO 330 I=1,L
330   IF(NEXT.EQ.TCYCLE(I))GO TO 350
      L=L+1
      TCYCLE(L)=NEXT
C     WRITE(6,610)TCYCLE(L)
      GO TO 325
350   CYCLE(1)=NEXT
      K=1
      DO 360 J=1,N
      IF((L-J+1).EQ.I)GO TO 379
      CYCLE(J+1)=TCYCLE(L-J+1)
      K=K+1
360   CONTINUE
379   L=K
      WRITE(6,*)(CYCLE(I),I=1,L)
      RETURN
      END
```

BIBLIOGRAPHY

1. Bazaraa, M. S. and J. J. Jarvis, Linear Programming and Network Flows, John Wiley & Sons, N.Y., 1977.

2. Bazaraa, M. S. and R. W. Langley, "A Dual Shortest Path Algorithm," SIAM Applied Mathematics, Vol. 26, No. 3 (1974), pp. 496-501.

3. Bellman, Richard, "On a Routing Problem," Quarterly Applied Mathematics, Vol. 16, No. 1 (1958), pp. 87-90.

4. Bennington, G. E., "A Simplex Algorithm for the Shortest Chain Problem," Report No. 72, North Carolina State University at Raleigh, 1971.

5. Bennington, G. E., "An Efficient Minimal Cost Flow Algorithm," Management Science, Vol. 19, No. 9 (1973), pp. 1042-1051.

6. Busacker, R. G. and T. L. Saaty, Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill Book Co., New York, 1965.

7. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, N.J., 1963.

8. Dijkstra, E. W., "A Note on Two Problems in Connection with Graphs," Numerische Mathematik, Vol. 1 (1959), pp. 269-271.

9. Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms," Operations Research, Vol. 17 (1969), pp. 395-412.

10. Dunn, O. J. and V. A. Clark, Applied Statistics: Analysis of Variance and Regression, John Wiley & Sons, N. Y., 1974.

11. Eisenhart, Churchill, "The Assumptions Underlying the Analysis of Variance," Biometrics, Vol. 3, No. 1 (1947), pp. 1-21.

12. Ferland, J. A., "Minimum Cost Multicommodity Circulation Problem with Convex Arc-Costs," Transportation Science, Vol. 8, No. 4 (1974), pp. 355-360.

13. Florian, M. and P. Robert, "A Direct Search Method to Locate Negative Cycles in a Graph," Management Science, Vol. 17, No. 5 (1971), pp. 307-310.

14. Florian, M. and P. Robert, Rejoinder: Direct Search Method for Finding Negative Cycles," Management Science, Vol. 19, No. 3 (1972), pp. 335-336.

15. Floyd, R. W., "Algorithm 97, Shortest Path," Comm. of the ACM, Vol. 15, No. 6 (1962), p. 345.

16. Ford, L. R. and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, 1962.

17. Frank, H. and I. Frisch, Communication, Transmission, and Transportation Networks, Addison-Wesley Company, Reading, Mass., 1971.

18. Golden, Bruce, "A Minimum-Cost Multicommodity Network Flow Problem Concerning Imports and Exports," Technical Report No. 105, Operations Research Center, Massachusetts Institute of Technology, 1974.

19. Golden, Bruce, "Shortest-Path Algorithms: A Comparison," Operations Research, Vol. 24, No. 6 (1976), pp. 1164-1168.

20. Hick, C. R., Fundamental Concepts in Design of Experiments, Holt, Rinehart and Winston, Inc., U.S.A., 1973.

21. Hines, W. W. and D. C. Montgomery, Probability and Statistics in Engineering and Management Science, The Ronald Press Co., N. Y., 1972.

22. Hu, T. C., "A Decomposition Algorithm for Shortest Paths in a Network," IBM Research Center Report, RC 1562, 1966.

23. Hu, T. C., Integer Programming and Network Flows, Addison Wesley, Reading, Mass., 1969.

24. Johnson, E. L., "Networks and Basic Solutions," Operation Research, Vol. 14 (1966), pp. 619-624.

25. Klein, M., "A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems," Management Science, Vol. 14, No. 3 (1967), pp. 205-220.

26. Klingman, D., Napier, A., Stutz, J., "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," Management Science, Vol. 20, No. 5 (1974), pp. 814-821.

27. Lindman, H. R., Analysis of Variance in Complex Experimental Designs, W. H. Freeman and Co., San Francisco, 1974.

28. Searle, S. R., Linear Models, John Wiley & Sons, N. Y., 1971.

29. Yen, J. Y., "An Algorithm for Finding Shortest Routes from all Source Nodes to a Given Destination in General Networks," Quarterly Applied Mathematics, Vol. 27, No. 4 (1970), pp. 526-530.

30. Yen, J. Y., "A Modified 1/2 N^3-steps Algorithm for Detecting Negative Loops or Finding all Shortest Routes from a Fixed Origin in N-node General Networks, Paper presented at the TIMS Tenth American Meeting, Atlanta, Georgia, Oct. 1-3, 1969.

31. Yen, J. Y., "On the Efficiency of a Direct Search Method to Locate Negative Cycles in a Network," Management Science, Vol. 19, No. 3 (1972), pp. 333-335.

32. Zaheh, L. A., Neustadt, L. W., Balakrishnan, A. V., Computing Methods in Optimization Problems-2, Academic Press, N. Y., 1969.